REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
24/May/2001	DISSERTATION	

4. TITLE AND SUBTITLE
ATTACK RESISTANT MOBILE AGENTS FOR INTRUSION DETECTION SYSTEMS

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
CAPT HUMPHRIES JEFFREY W

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
TEXAS A&M UNIVERSITY

8. PERFORMING ORGANIZATION REPORT NUMBER
CI01-80

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
THE DEPARTMENT OF THE AIR FORCE
AFIT/CIA, BLDG 125
2950 P STREET
WPAFB OH 45433

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Unlimited distribution
In Accordance With AFI 35-205/AFIT Sup 1

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE		147	19b. TELEPHONE NUMBER *(Include area code)*

ATTACK RESISTANT MOBILE AGENTS FOR INTRUSION DETECTION SYSTEMS

JEFFREY WAYNE HUMPHRIES
Captain
United States Air Force

2001

147 Pages

DOCTOR OF PHILOSOPHY

Texas A&M University

ABSTRACT

The rapid increase in attacks on computer systems has made intrusion detection systems (IDSs) increasingly popular. An emerging research area involves using mobile agents in implementing such systems. The lack of security for mobile agents is a primary factor that has inhibited their widespread use in real-world applications, including intrusion detection systems. Thus, providing security for mobile agents is key to building useful applications based on the mobile agent paradigm.

The core problem of such an agent-based system is this: an agent's owner cannot trust its agent, and mobile agents and their hosts do not trust each other. Worse still, if a host is penetrated and the attacker gains access to a traveling agent, he will potentially be given a wealth of new information that will help him attack and further penetrate the system. If an attacker can obtain detailed knowledge of the detection systems installed at a particular site, he will be better able to avoid its triggers. Hence, security for these agents is critical.

The overall intent of this research is to develop a methodology for protecting mobile agents in intrusion detection systems and to demonstrate the ability of such agents to address the shortcomings in current host-based systems. This methodology will support the defense of computer systems through a secure, mobile agent-based architecture. In support of this research, a secure mobile agent IDS prototype was created. The capabilities of this prototype as well as experimental results are described.

BIBLIOGRAPHY

[1] W. Jansen, P. Mell, T. Karygiannis, and D. Marks, "Applying Mobile Agents to Intrusion Detection and Response," NIST Interim Report (IR) 6416, National Institute of Standards and Technology, Computer Security Division, Gaithersburg, MD, October, 1999.

[2] R. Heady, G. Luger, A. Maccabe, and M. Servilla, "The Architecture of a Network Level Intrusion Detection System," Technical Report CS90-20, Department of Computer Science, University of New Mexico, Albuquerque, NM, August, 1990.

[3] B. Mukherjee, L. T. Heberlein, and K. N. Levitt, "Network Intrusion Detection," IEEE Network, vol. 8, no. 3, March, 1994, pp. 26-41.

[4] M. Crosbie and G. Spafford, "Active Defense of a Computer System Using Autonomous Agents," Technical Report CSD-TR-95-008, COAST Group, Department of Computer Sciences, Purdue University, West Lafayette, IN, February, 1995.

[5] N. M. Karnik and A. R. Tripathi, "Design Issues in Mobile Agent Programming Systems," IEEE Concurrency, vol. 6, no. 3, July-September, 1998, pp. 52-61.

[6] D. Chess, B. Grosof, C. Harrison, D. Levine, C. Parris, and G. Tsudik, "Itinerant Agents for Mobile Computing," IBM Research Report RC 20010, IBM Research Division, T.J. Watson Research Center, Yorktown Heights, NY, March 27, 1995.

[7] K. Rothermel, F. Hohl, and N. Radouniklis, "Mobile Agent Systems: What Is Missing?," in Proc. International Working Conference on Distributed Applications and Interoperable Systems (DAIS'97), Cottbus, Germany, September 30, 1997, pp. 111-124.

[8] T. Chia and S. Kannapan, "Strategically Mobile Agents," in Proc. First International Workshop on Mobile Agents, vol. 1219, Lecture Notes in Computer Science, K. Rothermel and R. Popsecu-Zeletin, Eds., Berlin, Germany: Springer-Verlag, 1997, pp. 1-10.

[9] D. Kotz and R. S. Gray, "Mobile Agents and the Future of the Internet," ACM Operating Systems Review, vol. 33, no. 3, August, 1999, pp. 7-13.

[10] D. B. Lange, "Mobile Objects and Mobile Agents: The Future of Distributed Computing?," in Proc. European Conference on Object-Oriented Programming, Brussels, Belgium, July 20-24, 1998, pp. 1-12.

[11] Y. Aridor and D. B. Lange, "Agent Design Patterns: Elements of Agent Application Design," in Proc. Second International Conference on Autonomous Agents, Minneapolis, MN, May 10-13, 1998, pp. 108-115.

[12] J. Bredin, D. Kotz, and D. Rus, "Market-based Resource Control for Mobile Agents," in Proc. Second International Conference on Autonomous Agents, Minneapolis, MN, May 10-13, 1998, pp. 197-204.

[13] M. O. Hofmann, A. McGovern, and K. R. Whitebread, "Mobile Agents on the Digital Battlefield," in Proc. Second International Conference on Autonomous Agents, Minneapolis, MN, 1998, pp. 219-225.

[14] A. Fuggetta, G. P. Picco, and G. Vigna, "Understanding Code Mobility," IEEE Transactions on Software Engineering, vol. 24, no. 5, May, 1998, pp. 342-361.

[15] N. Minar, K. H. Kramer, and P. Maes, "Cooperating Mobile Agents for Mapping Networks," in Proc. First Hungarian National Conference on Agent Based Computing, Budapest, Hungary, May, 1998, p. 12.

[16] W. R. Cockayne and M. Zyda, Mobile Agents, Greenwich, CT: Manning Publications Co., 1998.

[17] C. Schramm, A. Bieszczad, and B. Pagurek, "Application-Oriented Network Modeling with Mobile Agents," in Proc. Network Operations and Management Symposium, New Orleans, Louisiana, February 15-20, 1998, pp. 696-700.

[18] T. Magedanz, K. Rothermel, and S. Krause, "Intelligent Agents: An Emerging Technology for Next Generation Telecommunications?," in Proc. Fifteenth Annual Joint Conference of the IEEE Computer Societies, San Francisco, California, March 24-28, 1996, pp. 464-472.

[19] M. Crosbie and G. Spafford, "Defending a Computer System Using Autonomous Agents," Technical Report CSD-TR-95-022, COAST Group, Department of Computer Sciences, Purdue University, West Lafayette, IN, 1995.

[20] L. Garber, "Denial-of-Service Attacks Rip the Internet," Computer, vol. 33, no. 4, April, 2000, pp. 12-17.

[21] "Computer Emergency Response Team," Available at http://www.cert.org, February, 2001.

[22] D. Schoder and T. Eymann, "The Real Challenges of Mobile Agents," Communications of the ACM, vol. 43, no. 6, June, 2000, pp. 111-112.

[23] L. L. Kassab and J. Voas, "Agent Trustworthiness," in Proc. ECOOP Workshop on Distributed Object Security and 4th Workshop on Mobile Object Systems: Secure Internet Mobile Computations, Brussels, Belgium, July 20-21, 1998, pp. 121-133.

[24] J. J. Ordille, "When Agents Roam, Who Can You Trust?," in Proc. First Annual Conference on Emerging Technologies and Applications in Communications, Portland, OR, May 7-10, 1996, pp. 188-191.

[25] J. Riordan and B. Schneier, "Environmental Key Generation Towards Clueless Agents," in Mobile Agents and Security, G. Vigna, ed., Berlin: Springer-Verlag, 1998, pp. 15-24.

[26] D. S. Milojicic, F. Douglis, and R. Wheeler, ed., Mobility: Processes, Computers, and Agents, Reading, MA: Addison-Wesley, 1999.

[27] S. Franklin and A. Graesser, "Is It an Agent, or Just a Program? A Taxonomy for Autonomous Agents," in Intelligent Agents III: Agent Theories, Architectures, and Languages, J. Mueller, ed., Berlin: Springer-Verlag, 1997.

[28] C. F. Tschudin, "Mobile Agent Security," in Intelligent Information Agents: Agent Based Information Discovery and Management on the Internet, M. Klusch, ed., Berlin, Germany: Springer-Verlag, 1999, pp. 431-446.

[29] J. Vitek and G. Castagna, "Mobile Computations and Hostile Hosts," in Proc. 10th Journées Francophones des Langages Applicatifs (JFLA), Avoriaz, France, January, 1999, p. 241.

[30] A. D. Rubin and D. E. Geer Jr., "Mobile Code Security," IEEE Internet Computing, vol. 2, no. 6, November-December, 1998, pp. 30-34.

[31] D. Hagimont and L. Ismail, "A Protection Scheme for Mobile Agents on Java," in Proc. Third Annual ACM/IEEE International Conference on Mobile Computing and Networking, Budapest, Hungary, September 26-30, 1997, pp. 215-222.

[32] B. S. Yee, "A Sanctuary for Mobile Agents," Technical Report CS97-537, Computer Science Department, University of California at San Diego, April 28, 1997.

[33] W. M. Farmer, J. D. Guttman, and V. Swarup, "Security for Mobile Agents: Issues and Requirements," in Proc. 19th National Information Systems Security Conference, Baltimore, MD, October 22-25, 1996, pp. 591-597.

[34] F. Hohl, "Time Limited Blackbox Security: Protecting Mobile Agents from Malicious Hosts," in Mobile Agents and Security, G. Vigna, ed., Berlin: Springer-Verlag, 1998, pp. 92-113.

[35] D. M. Chess, "Security Issues in Mobile Code Systems," in Mobile Agents and Security, G. Vigna, ed., Berlin: Springer-Verlag, 1998, pp. 1-14.

[36] G. Vigna, "Cryptographic Traces for Mobile Agents," in Mobile Agents and Security, G. Vigna, ed., Berlin: Springer-Verlag, 1998, pp. 137-153.

[37] L. L. Kassab and J. Voas, "Towards Fault-Tolerant Mobile Agents," in Proc. Distributed Computing on the Web Workshop (DCW '98), Rostock, Germany, June, 1998, pp. 96-106.

[38] W. Jansen and T. Karygiannis, "Mobile Agent Security," NIST Special Publication 800-19, National Institute of Standards and Technology, Computer Security Division, Gaithersburg, MD, August, 1999.

[39] T. Sander and C. F. Tschudin, "Protecting Mobile Agents Against Malicious Hosts," in Mobile Agents and Security, G. Vigna, ed., Berlin: Springer-Verlag, 1998, pp. 44-60.

[40] T. Sander and C. F. Tschudin, "On Software Protection via Function Hiding," in Proc. Second International Workshop on Information Hiding, Portland, OR, April 15-17, 1998, pp. 111-123.

[41] N. M. Karnik and A. R. Tripath, "Security in the Ajanta Mobile Agent System," Technical Report RZ 2996, Department of Computer Science, University of Minnesota, Minneapolis, MN, May, 1999.

[42] M. Bellare and B. S. Yee, "Forward Integrity for Secure Audit Logs," Available at http://www.cs.ucsd.edu/~bsy/pub/fi.ps, February 9, 2000.

[43] K. Smith and R. Paranjape, "Mobile Agents for Web-based Medical Image Retrieval," in Proc. 1999 IEEE Canadian Conference on Electrical and Computer Engineering, Edmonton, Alberta, Canada, May 9-12, 1999, pp. 966-970.

[44] D. Rus, R. Gray, and D. Kotz, "Autonomous and Adaptive Agents that Gather Information," in Proc. AAAI '96 International Workshop on Intelligent Adaptive Agents, Portland, OR, August, 1996, pp. 107-116.

[45] B. Schneier and J. Kelsey, "Secure Audit Logs to Support Computer Forensics," ACM Transactions on Information and System Security, vol. 2, no. 2, May, 1999, pp. 159-176.

[46] K. Neuenhofen and M. Thompson, "A Secure Marketplace for Mobile Java Agents," in Proc. Second International Conference on Autonomous Agents, Minneapolis, MN, May 10-13, 1998, pp. 212-218.

[47] J. Baek, "A Design of a Protocol for Detecting a Mobile Agent Clone and Its Correctness Proof Using Coloured Petri Nets," Technical Report TR-DIC-CSL-1998-002, Department of Information and Communications, Kwangju Institute of Science and Technology, Kwangju, Republic of Korea, 1998.

[48] F. B. Schneider, "Towards Fault-Tolerant and Secure Agentry," in Proc. 3rd ECOOP Workshop on Mobile Object Systems, Jyvalskyla, Finland, June, 1997, pp. 1-14.

[49] J. E. White, "Telescript Technology: Mobile Agents," in Software Agents, J. M. Bradshaw, ed., Menlo Park, CA: AAAI/MIT Press, 1997, pp. 437-472.

[50] J. Tardo and L. Valente, "Mobile Agent Security and Telescript," in Proc. IEEE COMPCON, Santa Clara, CA, February 25-28, 1996, pp. 58-63.

[51] R. S. Gray, D. Kotz, G. Cybenko, and D. Rus, "D'Agents: Security in a Multiple-Language, Mobile-Agent System," in Mobile Agents and Security, G. Vigna, ed., Berlin: Springer-Verlag, 1998, pp. 154-187.

[52] D. Kotz, R. Gray, S. Nog, D. Rus, S. Chawala, and G. Cybenko, "AGENT TCL: Targeting the Needs of Mobile Computers," IEEE Internet Computing, vol. 1, no. 4, July-August, 1997, pp. 58-67.

[53] T. Walsh, N. Paciorek, and D. Wong, "Security and Reliability in Concordia," in Mobility: Processes, Computers, and Agents, D. Milojicic, F. Douglis, and R. Wheeler, eds., Reading, MA: Addison-Wesley, 1999, pp. 525-534.

[54] J. Baumann, F. Hohl, K. Rothermel, and M. Straser, "Mole - Concepts of a Mobile Agent System," World Wide Web, vol. 1, no. 3, July-September, 1998, pp. 123-137.

[55] H. Peine and T. Stolpmann, "The Architecture of the Ara Platform for Mobile Agents," in Proceedings of the First International Workshop on Mobile Agents, K. Rothermel and R. Popescu-Zeletin, eds., Berlin: Springer-Verlag, 1997, pp. 50-61.

[56] G. Karjoth, D. B. Lange, and M. Oshima, "A Security Model for Aglets," in Mobile Agents and Security, G. Vigna, ed., Berlin: Springer-Verlag, 1998, pp. 188-205.

[57] D. Johansen, R. van Renesse, and F. B. Schneider, "Operating System Support for Mobile Agents," in Proc. 5th Workshop on Hot Topics in Operating Systems, Orcas Island, WA, May 4-5, 1995, pp. 42-45.

[58] T. Sandholm and Q. Huai, "Nomad: Mobile Agent System for an Internet-Based Auction House," IEEE Internet Computing, vol. 4, no. 2, March-April, 2000, pp. 80-86.

[59] D. Wong, N. Paciorek, T. Walsh, J. DiCelie, M. Young, and B. Peet, "Concordia: An Instructure for Collaborating Mobile Agents," in First International Workshop on Mobile Agents, vol. 1219, Lecture Notes in Computer Science, Berlin: Springer-Verlag, 1997, pp. 86-97.

[60] A. O. Freier, P. Karlton, and P. C. Kocher, "The SSL Protocol, Version 3.0," Available at http://home.netscape.com/eng/ssl3/ssl-toc.html, May 11, 2000.

[61] J. Arthursson, J. Engblom, I. Jonsson, R. Mirza, G. Naeser, M. Olsson, R. Ottenhag, D. Sahlin, M. Schmid, B. Spolander, and E. Zolfonoon, "A Platform for Secure Mobile Agents," in Proc. Second International Conference and Exhibition

on the Practical Aplication of Intelligent Agents and Multi-Agent Technology, London, England, April, 1997, pp. 109-120.

[62] G. Cabri, L. Leonardi, and F. Zambonelli, "Mobile Agent Technology: Current Trends and Perspectives," in Proc. Associazione Italiana per l'Informatica ed il Calcolo Automatico (AICA'98), Naples, Italy, November, 1998, pp. 1-12.

[63] P. E. Proctor, The Practical Intrusion Detection Handbook, Upper Saddle River, NJ: Prentice Hall, 2001.

[64] H. Debar, M. Dacier, and A. Wespi, "Towards a Taxonomy of Intrusion-Detection Systems," Technical Report RZ 3030, IBM Research Division, Zurich Research Laboratory, Zurich, Switzerland, June, 1998.

[65] G. B. White, E. A. Fisch, and U. W. Pooch, "Cooperating Security Managers: A Peer-Based Intrusion Detection System," IEEE Network, vol. 10, no. 1, January-February, 1996, pp. 20-23.

[66] D. S. Alberts, "The Unintended Consequences of Information Age Technologies," Available at http://www.ndu.edu/ndu/inss/books/uc/uchome.html, December, 2000.

[67] D. E. Denning, "Protection and Defense of Intrusion," Available at http://www.cosc.georgetown.edu/%7edenning/infosec/USAFA.html, December, 2000.

[68] J. S. Balasubramaniyan, J. O. Garcia-Fernandez, D. Isacoff, E. Spafford, and D. Zamboni, "An Architecture for Intrusion Detection Using Autonomous Agents," COAST Technical Report 98/05, COAST Laboratory, Purdue University, West Lafayette, IN, June 11, 1998.

[69] D. Frincke, D. Tobin, J. McConnell, J. Marconi, and D. Polla, "A Framework for Cooperative Intrusion Detection," in Proc. 21st National Information Systems Security Conference, Arlington, VA, October, 1998, pp. 361-373.

[70] J. Evans and D. Frincke, "Trust Mechanisms for Hummingbird," Available at http://www.acm.org/crossroads/xrds2-4/humming.html, March 30, 2000.

[71] B. C. Neuman and T. Tso, "Kerberos: An Authentication Service for Computer Networks," IEEE Communications, vol. 32, no. 9, September, 1994, pp. 33-38.

[72] G. G. Helmer, J. S. K. Wong, V. Honavar, and L. Miller, "Intelligent Agents for Intrusion Detection," in Proc. IEEE Information Technology Conference, Syracuse, NY, September, 1998, pp. 121-124.

[73] M. Asaka, S. Okazawa, A. Taguchi, and S. Goto, "A Method of Tracing Intruders by Use of Mobile Agents," in Proc. 9th Annual Internetworking Conference (INET'99), San Jose, CA, June, 1999, pp. 1-12.

[74] P. A. Porras and P. G. Neumann, "EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances," in Proc. Nineteenth National Information Systems Security Conference, Baltimore, MD, 1997, pp. 353-365.

[75] B. Schneier, Applied Cryptography, Second Edition, New York: John Wiley & Sons, Inc., 1996.

[76] A. J. Menezes, P. C. van Oorshot, and S. A. Vanstone, Handbook of Applied Cryptography, Boca Raton, FL: CRC Press, 1997.

[77] W. Diffie and M. E. Hellman, "New Directions in Cryptography," IEEE Transactions on Information Theory, vol. IT-22, no. 6, November, 1976, pp. 644-654.

[78] W. Stallings, Network and Internetwork Security: Principles and Practice, Englewood Cliffs, NJ: Prentice Hall, 1995.

[79] A. Corradi, M. Cremonini, and C. Stefanelli, "Locality Abstractions and Security Models in a Mobile Agent Environment," in Proc. Seventh IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, Los Alamitos, CA, June 17-19, 1998, pp. 230-235.

[80] H. Vogler, T. Kunkelmann, and M.-L. Moschgath, "An Approach for Mobile Agent Security and Fault Tolerance Using Distributed Transactions," in Proc. International Conference on Parallel and Distributed Systems, Seoul, Korea, December 10-13, 1997, pp. 268-274.

[81] M. S. Greenberg, J. C. Byington, and D. G. Harper, "Mobile Agents and Security," IEEE Communications Magazine, vol. 36, no. 7, July, 1998, pp. 76-85.

[82] T. Thorn, "Programming Languages for Mobile Code," ACM Computing Surveys, vol. 29, no. 3, September, 1997, pp. 213-239.

[83] P. Felber, R. Guerraoui, and M. E. Fayad, "Putting OO Distributed Programming to Work," Communications of the ACM, vol. 42, no. 11, November, 1999, pp. 97-101.

[84] M. Blaze, J. Feigenbaum, and J. Lacy, "Decentralized Trust Management," in Proc. 1996 IEEE Symposium on Security and Privacy, Oakland, CA, May 6-8, 1996, pp. 164-173.

[85] J. Kay, J. Etzl, G. Rao, and J. Thies, "The ATL Postmaster: A System for Agent Collaboration and Information Dissemination," in Proc. Second International Conference on Autonomous Agents, Minneapolis, MN, May 9-13, 1998.

[86] J. Fiedler, "A Distributed Personalized News System Based on Mobile Agents," in Proc. 36th Annual ACM Southeast Conference, Marietta, GA, April 1-3, 1998, pp. 130-135.

[87] HostSentry, Psionic Software Inc., Available at http://www.psionic.com/, 2000.

[88] PortSentry, Psionic Software Inc., Available at http://www.psionic.com/, 2000.

[89] LogCheck, Psionic Software Inc., Available at http://www.psionic.com/, 2000.

[90] Linux Intrusion Detection System, LIDS.org, Available at http://www.lids.org, 2000.

[91] OpenWall, OpenWall Project, Available at http://www.openwall.com, 2000.

[92] NetSaint Network Monitor, Available at http://netsaint.sourceforge.net, 2000.

[93] CyberCop Monitor, PGP Security, Available at http://www.pgp.com/products/cybercop-monitor/default.asp, 2000.

[94] Dragon Sensor, Network Security Wizards, Inc., Available at http://www.securitywizards.com/intro.html, 2000.

[95] Dragon Squire, Network Security Wizards, Inc., Available at http://www.securitywizards.com/intro.html, 2000.

[96] Patriot IDS, Patriot Technologies, Available at http://patriot-tech.com/ids.htm, 2000.

[97] PreCis Security Toolkit, PRC PreCis, Available at http://www.bellevue.prc.com/precis/, 2000.

[98] Sygate Enterprise Network, Sygate Technologies, Inc., Available at http://www.sygate.com/products/sms_ov.htm, 2000.

ATTACK RESISTANT MOBILE AGENTS FOR INTRUSION DETECTION SYSTEMS

A Dissertation

by

JEFFREY WAYNE HUMPHRIES

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

May 2001

Major Subject: Computer Science

ATTACK RESISTANT MOBILE AGENTS FOR INTRUSION DETECTION SYSTEMS

A Dissertation

by

JEFFREY WAYNE HUMPHRIES

Submitted to Texas A&M University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Approved as to style and content by:

Udo W. Pooch
(Chair of Committee)

John Yen
(Member)

Michael T. Longnecker
(Member)

Riccardo Bettati
(Member)

Wei Zhao
(Head of Department)

May 2001

Major Subject: Computer Science

ABSTRACT

Attack Resistant Mobile Agents for Intrusion Detection Systems. (May 2001)

Jeffrey Wayne Humphries, B.S., United States Air Force Academy;

M.S., Georgia Institute of Technology

Chair of Advisory Committee: Dr. Udo W. Pooch

The rapid increase in attacks on computer systems has made intrusion detection systems (IDSs) increasingly popular in academic, corporate, and government networks. While IDSs are not new, research into improving their performance is ongoing. One new area involves using mobile agents in implementing intrusion detection systems. Research in this area is in its infancy and has not yet entered the mainstream community. The lack of security for mobile agents is a primary factor that has inhibited their widespread use in real-world applications, including intrusion detection systems. Thus, providing security for mobile agents is key to building useful applications based on the mobile agent paradigm.

Before these systems can be deployed in real settings, the major obstacle of providing adequate security for the agents themselves must be overcome. The core problem of such an agent-based system is this: an agent's owner cannot trust its agent, and agents and host systems do not trust each other. Worse still, if a host is penetrated and the attacker gains access to a traveling agent, he will potentially be given a wealth of new

information that will help him attack other hosts in the network and further penetrate the system. If an attacker can obtain detailed knowledge of the detection systems installed at a particular site, he will be better able to avoid its triggers. Hence, security for these agents is critical. Unfortunately, solutions to many of these problems do not currently exist.

The overall intent of this research is to develop a methodology for protecting mobile agents in intrusion detection systems and to demonstrate the ability of such agents to address the shortcomings in current host-based systems. This methodology will support the defense of computer systems through a secure, mobile agent-based architecture. In support of this research, a secure mobile agent IDS prototype was created. The capabilities of this prototype as well as experimental results are described.

DEDICATION

"…Let us run with perseverance the race marked out for us." – Hebrews 12:1

I dedicate this dissertation to my loving and gracious wife, Erin, who runs the race by my side.

ACKNOWLEDGMENTS

I would like to thank Dr. Udo Pooch for his guidance and leadership during this research. He has been an excellent mentor and guide. His management style gave me great latitude and freedom to explore the possibilities and productively use my time to accomplish this work.

I would also like to thank the other members of my committee, Dr. John Yen, Dr. Riccardo Bettati, Dr. Mike Longnecker, and Dr. Dickson Varner who have all helped me make this research stronger. Both inside the classroom and out, these professors have imparted knowledge, ideas, and skills that have enabled me to successfully complete this research. I thank them for their suggestions and encouragement.

I thank my parents, Buddy and Debbie, who helped me buy my first computer at a young age. They have always encouraged me to strive in my academic pursuits and have given me great freedom to explore my interests. More importantly, they raised me in a loving home where I learned the meaning of character and hard work.

I would also like to acknowledge the wonderful support of my wife, Erin. She has been a great encouragement to me during this process. She is the consummate helpmeet.

Finally, I would like to acknowledge the tremendous help of many graduate students with whom I had the pleasure of working and collaborating. Among the many who have helped me succeed at Texas A&M are John "Buck" Surdu, John Hill, Curt Carver, Mike Miller, Dan Ragsdale, and Jim Vaglia. I thank them all!

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

FIGURE Page

CHAPTER I

INTRODUCTION

A. Motivation

The rapid increase in attacks on computer systems has made intrusion detection systems (IDSs) increasingly popular in academic, corporate, and government networks. It is estimated that approximately $100 million dollars in sales of IDS systems were made in 1998 [1]. While IDSs are not new, research into improving their performance is ongoing. One new area involves using mobile agents in implementing intrusion detection systems. Research in this area is continuing in many universities and labs, but has not yet entered the mainstream community. The lack of security for mobile agents is a primary factor that has inhibited their widespread use in real-world applications, including intrusion detection systems. Thus, providing security for mobile agents is key to building useful applications based on the mobile agent paradigm.

An intrusion can be defined as "any set of actions that attempt to compromise the integrity, confidentiality, or availability of a resource" [2]. Thus, an intrusion detection system must identify "unauthorized use, misuse, and abuse of computer systems" [3] in a network. A good IDS should run independently of the processes already on a system, and should quickly provide information to a system administrator regarding any

---

This dissertation follows the style and format of *IEEE Transactions on Automatic Control.*

suspicious activity on hosts in the network [4]. Also, an IDS should be well protected and resilient to subversion, as it could be the first target of any attack to a system.

A mobile agent is simply "a program that represents a user in a computer network and can migrate autonomously from node to node, to perform some task on behalf of the user" [5]. Mobile agents offer several advantages over the more classic client-server and peer-to-peer paradigms [6-12]. First, when implemented correctly, they can reduce the overall communication traffic in the network. Because a mobile agent doesn't always have to stay in contact with its originating host, the number of interactions to carry out some action can be reduced. Second, mobile agents have the ability to engage in high-bandwidth communication with a server. Since the code that makes up a mobile agent is generally much smaller than the data to be operated on, moving the agent instead of the data can greatly reduce the communications load among hosts on the network [13]. Third, mobile agents allow the user to create specialized services by tailoring agents to a specific need. Service customization is a major asset provided by code mobility because re-tailored agents can be quickly designed to perform new functions [14-18].

Using mobile agents in the intrusion detection domain offers several advantages over more traditional methods. Crosbie and Spafford [19] compared these small, lightweight agents to current monolithic approaches for IDSs and asserted several advantages. First, agents are more easily tailored and can be quickly changed to observe new host behaviors. Security experts recommend that networked organizations should regularly update their security systems, such as anti-virus and intrusion-detection software [20]. As organizations such as the Computer Emergency Response Team (CERT) [21] and

others find and disseminate information on new vulnerabilities, new agents can be written and dispatched to look for these problems on network hosts. Second, agents can be very efficient if they are written to be simple and to consume as few system resources as possible. These agents can impose a lower overhead on network bandwidth and other resources than can some of the current, more centralized approaches to intrusion detection. Third, intrusion detection systems using mobile agents are more fault tolerant in that the components that makeup the system are moving and are not centrally located. The failure of a host does not necessarily interfere with a mobile agent as it can easily reroute itself around problems in the network. This makes such systems more resilient to subversion. A final advantage of using mobile agents in this domain is that they scale well to larger systems. As a network grows larger, more agents can be added to migrate through hosts looking for suspicious activity.

It is clear that using mobile agents for intrusion detection offers many benefits over traditional approaches; however, before these systems can be deployed in real settings, the major obstacle of providing adequate security for the agents themselves must be overcome [22]. The core problem of such an agent-based system is this: an agent owner cannot trust its own agents (once they have been dispatched), and agents and the hosts they run on do not trust each other (mutual suspicion) [23, 24]. Even in a closed, so-called "trusted" environment, these problems are not easily overcome, because there is no guarantee that hosts and users are (or will remain) benevolent and cooperative [23]. Worse still, if a host is penetrated and the attacker gains access to a traveling agent, he will potentially be given a wealth of new information that will help him attack other

hosts in the network and further penetrate the system. "If an attacker has detailed knowledge of the detection systems installed at a particular site, he is better able to avoid its triggers. As such, it would be better to deploy an IDS whose triggers are not easily analyzable" [25]. Hence, security for these agents is critical. Chess, et al. summarize this problem well: "It is difficult to exaggerate the value and importance of security in an itinerant agent environment. It is, without a doubt, one of the cornerstone issues" [6]. Unfortunately, solutions to many of these problems currently do not exist.

B. Research Objectives

The overall intent of this research was to develop a methodology for protecting mobile agents in distributed intrusion detection systems and to demonstrate the ability of such agents to address the shortcomings in current host-based IDSs. This methodology supports the defense of computer systems through a secure, mobile agent-based architecture. The following research objectives accomplished this intent:

- Research, develop, and apply mechanisms for securing mobile agents in intrusion detection systems that:
    - Ensure agent confidentiality, integrity, and availability through the use of cryptographic methods.
        - The confidentiality of mobile agent code and data will be protected during transit between hosts.
        - The integrity of mobile agent code will be monitored so that any changes to an agent will be detected.

        - The integrity and confidentiality of data collected during an agent's lifetime will be ensured so that subsequent hosts cannot view this data or make changes without detection.
        - Unauthorized changes to an agent's itinerary will be detected.
    - Protect mobile agents from denial-of-service attacks.
        - Malicious or malfunctioning hosts that attempt to remove or suspend agents will be detected.
        - Various hold-back, checkpointing, and other techniques will be used to ensure that agents are not removed from the system except by the originator.
    - Provide authentication of both agents and hosts to prevent spoofing.
- Measure and compare the capabilities of secure mobile agents.
- Develop a prototype secure mobile agent system to demonstrate the feasibility of this approach.

C. Overview

Chapter II presents a survey and review of the current literature that serves as background for this research. The purpose of Chapter II is to describe work that has been previously done and to provide sufficient background material for this dissertation. The domain areas described include mobile agents, intrusion detection, distributed systems, computer security, and cryptography.

Chapter III describes the proposed methodology for protecting mobile agents in the intrusion detection domain. This chapter describes the design of each component used in this methodology, with emphasis on the components that have been prototyped.

Chapter IV provides details of the actual implemented prototype system, including the mobile agent infrastructure, the security components, and the algorithms used. It also describes the interactions of the system components and how they support the methodology.

Chapter V describes the results of this doctoral research. It describes the performance of the overall system and the details of the testing procedures. This chapter also describes the experiments used to demonstrate the correctness of the methodology.

Chapter VI gives a summary and presents the major conclusions of this research. This chapter also summarizes the contributions of this work to the field of computer science and makes recommendations for future research.

CHAPTER II

LITERATURE REVIEW

A. Overview

The development of an architecture to secure mobile agents in intrusion detection systems borrows ideas from previous work in computer security, mobile agents, intrusion detection systems, and distributed systems. While there has been significant research in intrusion detection and mobile agents, there has been limited research in providing adequate, integrated security for mobile agents in the intrusion detection domain that provides agent confidentiality, integrity, and availability. The following sections survey significant previous work in these areas and discusses these areas in sufficient detail to provide background for this research.

B. Mobile Agents

Three basic programming paradigms exist for distributed computing [10]. In the client-server paradigm, a server provides some set of services that allows various clients in the network to access some system resources as illustrated in Figure 1a. The code that provides these services is located locally on the server; that is, the server holds the "know-how" for implementing the services as well as the processor capability to carry them out [10].

In the code-on-demand paradigm, the client holds the resources needed for execution, as well as the processing capability, but may lack the "know-how" to execute its task

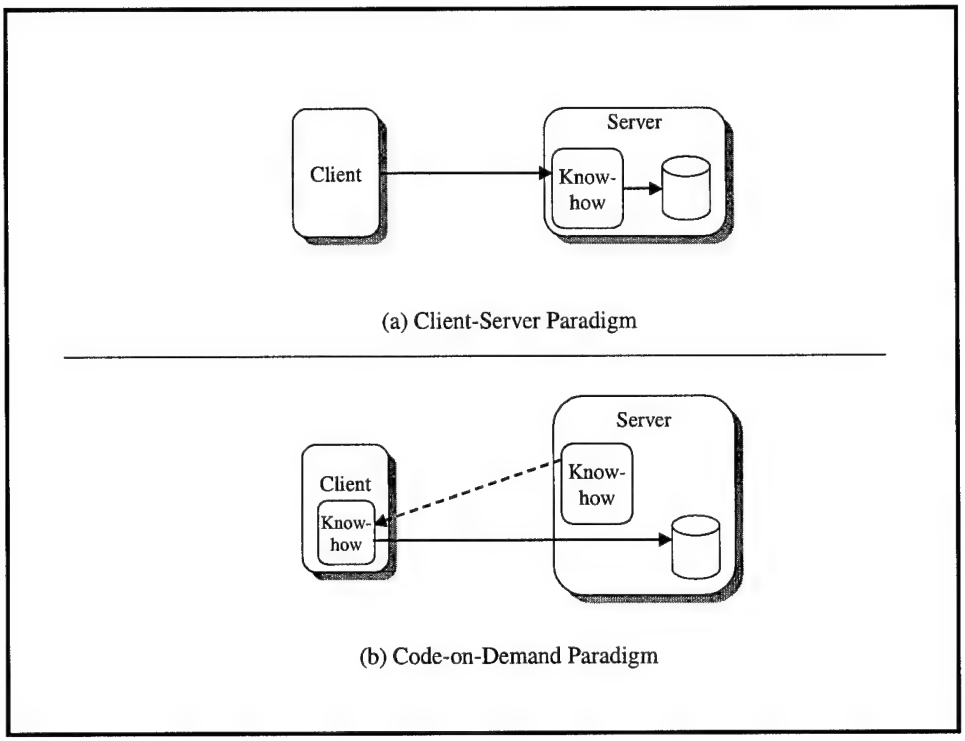

Figure 1: Client-server and code-on-demand paradigms [10].

due to a lack of code [10]. In these instances, another host in the network provides the needed code in order to allow the client to proceed with its execution (see Figure 1b). An example of this paradigm is a Java applet.

A third programming paradigm is that of the mobile agent. In this paradigm, hosts in the network are allowed flexibility to possess any mixture of know-how, access to resources, and processing capability [10]. The know-how is not tied to any particular host in the network, but is available to any host on the network through mobile agents [10]. This paradigm offers much greater flexibility in determining how a network's resources and processing capabilities will be utilized. The mobile agent paradigm is illustrated in Figure 2.

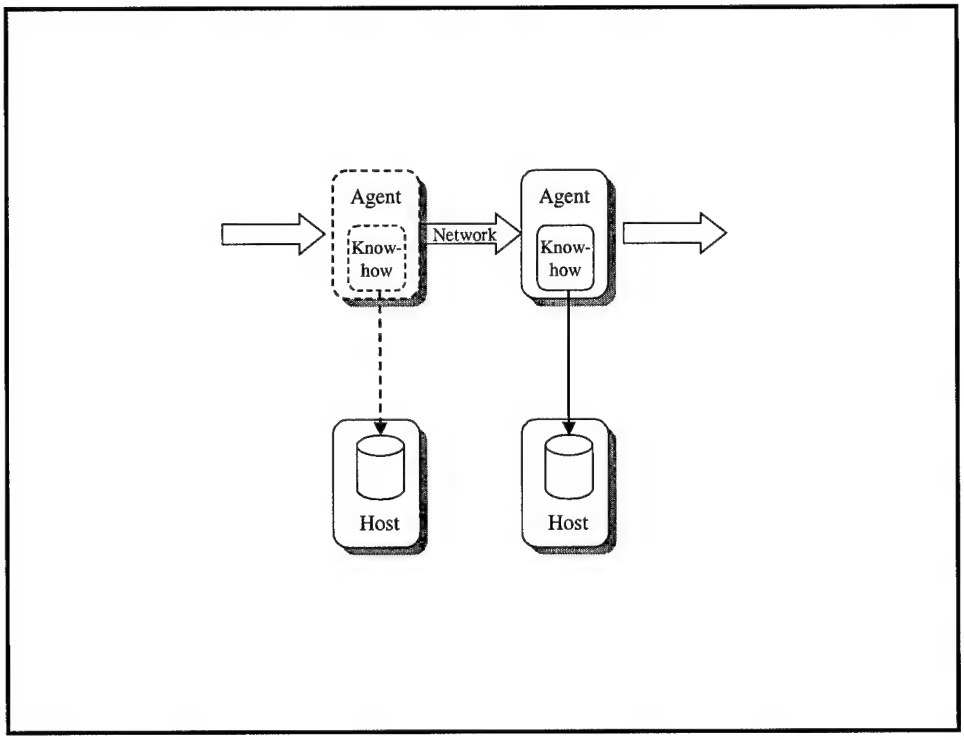

Figure 2: Basic mobile-agent paradigm [10].

The mobile agent paradigm is not new. The notion of a mobile agent developed from process migration and remote job processing research in the 1970's and mobile computing and distributed system research in the 1980's [18]. The pervasive use of computers and ubiquitous network interconnectivity, however, has effected a new influx in research of agents. This research falls within one of two camps: multi-agent/ intelligent systems and mobile agents in distributed systems. The former group of researchers focuses primarily on stationary objects that communicate with each other in a distributed system in order to perform some common task [26]. The latter group is doing research in areas of agent mobility; that is, the ability to autonomously transfer code and data between multiple hosts. While the characteristics of what exactly defines

a mobile agent vary throughout the literature [27], for purposes of clarity, a mobile agent will be defined for this research simply as "a program that represents a user in a computer network and can migrate autonomously from node to node, to perform some computation on behalf of the user" [5].

1. Mobile Agent Security

There are several vulnerabilities specific to mobile agent systems [23]. First, they have no inherent privacy – their code and data can be viewed by any host that executes it. Because an agent must rely on an execution environment in order to run, they have to expose themselves at some level to the system. This asymmetric power relationship between hosts and agents make agents very vulnerable [28, 29].

Second, a host can tamper with an agent's code or data. Just because a host can view an agent's data does not mean that it will necessarily change it. Malicious servers, however, can tamper with agent code or data in order to change its behavior. Clever servers can change an agent to actually do something malicious on its behalf. Malicious servers could even make random changes to an agent, knowing that those changes will alter the results it obtains or cause it to malfunction in some way.

Third, a malicious host can execute an agent's code incorrectly or incompletely. There is no guarantee that a host will execute an agent's code to completion, or that it will execute it at all. A host could execute only selective portions of an agent's code, bypassing critical subroutines, in order to cause an agent to obtain incorrect or incomplete results.

Fourth, an agent can be interrupted or terminated. A server could simply delete an agent that it currently hosts. If no mechanisms exist to counter this, the agent would be permanently lost. Even benevolent servers might inadvertently delete an agent if they are malfunctioning or experience a power loss.

Fifth, a host can lie about execution results, its own identity, or any other data gathered while an agent is on its site. Malicious hosts could give an agent incorrect data, knowing that this data will cause the system to incorrectly evaluate some result.

Sixth, a host can eavesdrop on any agent communication. Servers that are hosting agents can intercept and read any communication the agent has with other agents or other servers while on its system. This makes the task of securing these communications very difficult.

The reciprocal problem of protecting hosts from malicious agents is well-researched, and many technologies exist to address this problem [30]. The most effective technology is the sandbox. The basic sandbox model of protection attempts to contain mobile code in a special environment such that it can cause no harm to the host outside of that environment. The sandbox accomplishes this by placing special restrictions on file system and network access. The most common implementation of the sandbox model is in the Java interpreter running inside most Internet browsers [30]. The Java approach, while not perfect, does an adequate job of protecting hosts from malicious agents when implemented correctly.

Despite all of these potential problems, mobile agents offer many advantages over the more classic client-server and peer-to-peer paradigms [6]. These advantages, however,

cannot be fully realized in real-world applications until the problem of providing adequate security for these mobile agents has been addressed [31]. The following sections survey significant previous work in the area of mobile agents security and discuss these areas in sufficient detail to provide sufficient background knowledge.

a. Trusted Environments

The most common approach for protecting agents is simply to restrict agent migration to trusted hosts [32]. In this approach, agents cannot move to hosts that are unknown or untrusted. The primary problem with this approach is that it is absolute, in that the system either fully trusts a host or it does not, and "it relies on blind-faith that all hosts and agents are consistently benign" [23]. This form of protection is not realistic because most real-world applications will require agent mobility among unknown or untrusted hosts in order to function [33], and it is not always clear whether a host is trusted in advance or not [34]. In addition, this approach severely reduces the number of hosts that can be visited. One of the primary advantages of using mobile agents is the freedom they allow in migrating through open and unknown networks. This advantage would be eliminated by assuming trusted hosts because agents could not migrate to unfamiliar or newly added hosts. Moreover, even if agents were restricted to fully trusted networks, nothing prevents these once trusted hosts from later being subverted.

b. Reputation Servers

Another approach to agent protection relies on the ability to detect when a host misbehaves. These systems suggest using reputation services [34, 35] that provide

information on the trustworthiness of principals, similar to that of the Better Business Bureau for business. Before an agent migrates to an unknown host, it looks up the host's current reputation based on its past behavior to determine if it is trustworthy. If the host appears reputable, the agent proceeds with the migration. On the occasion that a host is found to be "misbehaving", it is reported to one of these services where it loses reputation because it has not "played by the rules".

This reputation approach suffers from several problems. First, it provides no real protection for the agent, in that only after something damaging has occurred will future agents be aware of potential problems. Second, hosts could be subject to "character assassination" attacks [34], where agents maliciously complain about a host to damage its reputation even though it has done nothing wrong. Third, it is not always possible to detect when a host has misbehaved, or even to classify the type or severity of damage inflicted by a malicious host.

c. Legal or Contractual Protection

Another non-technical approach to protecting agents is through legal or contractual means [32]. Agent servers promise, usually through some contractual arrangement, that they will not violate the privacy of agent code or data and that they will not interfere with an agent's computation [32]. This approach requires no additional technical mechanisms in order to secure agents as they migrate.

These arrangements, however, suffer from several major problems [32]. First, an agent owner must be able to detect when a host has breached its contract in order for this system to be feasible. This is difficult to do without additional technical mechanisms

included with each agent. Second, even if an owner could detect that an agent contract was violated, it must be able to prove through some non-repudiable evidence which host violated the contract. Third, agent owners must have some legal or other remedy available in which to penalize hosts that tamper with agents. This is difficult to do across administrative domains where no legal protections exist.

d. Execution Tracing

Vigna [36] suggests an approach that allows an agent's owner to detect if an agent has been modified and to prove which host performed the act. Vigna has proposed cryptographic traces of all computations performed by a mobile agent as it visits each host. The execution of an agent is recorded in such a manner that it cannot be forged by the host [36]. Each host must maintain logs of the operations performed by each agent while resident. Using cryptographic hashes of these traces, agent owners who suspect foul-play can request these logs which can verify whether an agent has been tampered with or not. The owner may then "use legal or organizational ways to get its damage refunded" [34].

This method of agent protection is exploratory and all "current approaches are very theoretical" [32]. In addition, manual verification of these logs "is an expensive process, which is likely to be inefficient and error-prone" [37]. In addition, the size and number of logs that must be maintained in a large system with many hosts and many agents is enormous [38]. These limitations greatly restrict the feasibility of this approach in any real-world mobile agent system.

e. Software Assertions

Software assertions are functions, usually evaluating to true or false, whose value is based on some semantic condition of a program's state. They have been used in the past in software testing, fault detection, and program correctness research. Software assertions have been proposed as a means of protecting agents by providing agent owners with snapshots of an agent's state as it migrates [23]. Using these snapshots, an agent owner can determine if an agent's results should be trusted.

Embedding protective assertions into agents is a complicated process. First, the agent owner has to employ some form of fault injection which attempts to identify weaknesses in an agent by simulating potential malicious behavior by a future host. The fault injection analysis then recommends which software assertions are best suited to protect the agent from discovered weaknesses [23]. Next, the agent owner must manually structure the software assertions for each agent and specify where the assertions should be embedded in the code. The owner must also specify what information is indicative of untrustworthy data returned from an agent. Following this process, a parsing tool inserts the assertions into the agent's source code. Finally, an analysis oracle is generated which runs on a specified machine and is responsible for collecting the results from agent assertions and analyzing the results. As an agent executes on a host, the results from each assertion is sent to the oracle for analysis. The oracle then informs the agent owner if it detected any unusual results.

The authors of this approach admit that it is not foolproof and describe several shortcomings [23]. First, software assertions still do not prevent a malicious host from

tampering with agents, removing assertions, or giving false information. Second, this approach involves additional overhead because all intermediate results must be shipped off to the oracle during each execution on every host. Another criticism is that the process of determining which assertions are needed and embedding them into the agents can be tedious and slow for a human operator.

f. Mobile Cryptography (Computing with Encrypted Functions)

Sander and Tschudin describe an approach to protect agent code from malicious hosts by using encrypted functions [39]. Their method uses encrypted programs that can be run on a foreign host without being decrypted [40]. As Figure 3 illustrates, this approach allows a host to execute an agent program embodying some encrypted function without the host being able to determine what the original function is. In other words, a function is transformed (encrypted) into another function which hides the original functionality, and this encrypted function is then implemented as a program that can be executed by an agent server [41]. An agent server can plainly see and execute these functions, but has no way of obtaining the original functionality.

So far, they have only been successful in using this approach with polynomial and rational functions. The major drawback is that no known methods exist in which arbitrary functions can be transformed into encrypted forms for use in mobile agents. Even if this did work, it would be a dubious prospect to require hosts to allow an agent to execute arbitrary encrypted functions. Also, while this technique could be powerful if

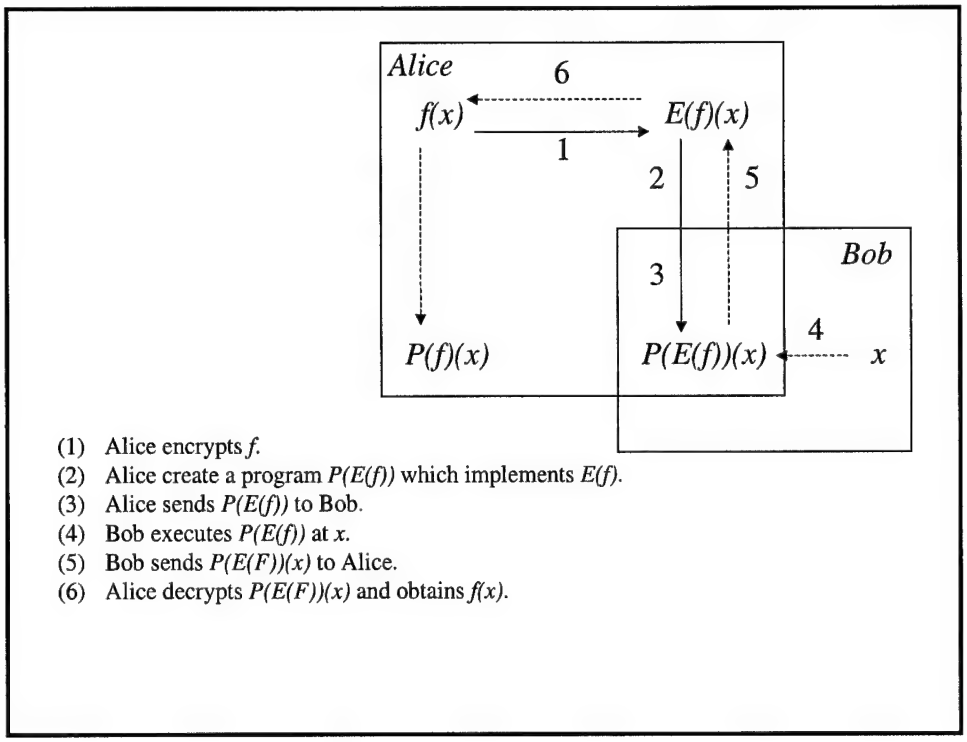

Figure 3: Computing with encrypted functions [39].

extended to use with arbitrary programs, agents are still vulnerable to denial of service attacks, replay, and experimental extraction [38].

g. Obfuscated Code

Hohl suggests an approach called "time limited blackbox protection" [34] in which agent code is scrambled (not encrypted) in such a way that it makes its function very difficult to understand by a third party (see Figure 4). Even though the code remains readable, its structure, variable names, and control flow are so jumbled, that it would take someone a long period of time to decipher what the program is doing. In Hohl's

approach, he suggests using this method when the code is not intended for long-term concealment, as obfuscation does not carry the same time protections as encryption does.

There are several problems with this approach. First, it is impossible to determine the relationship between the amount of code scrambling and the amount of time protection it affords. This is especially important as many attacks are automated and a computer would require much less analysis time than a human. Second, there are no known algorithms that can take arbitrary code and scramble it for time protection of a given length [38]. Thus, it is impossible to say that a given amount of obfuscation results in a specified amount of protection.

```
if (a(b) < c) {
    b = s(d(e) + f);
}
```

Original Code

Obfuscated Code

```
z = 0;
DO
  if (z=0) then t1=a(b);z=1;continue;
  if (z=1) then t2=t1<c;z=2;continue;
  if (t2) then t3=d(e);z=3;continue;
  if (z=3) then t4=t3+f;z=4;continue;
  if (z=4) then b=t4;z=5;continue;
  if (z=5) then break;
LOOP
```

Figure 4: Code obfuscation [34].

h. Partial Result Encapsulation

Another approach used to protect agents is called partial result encapsulation, where results of an agent's actions are cryptographically encapsulated before leaving each host. When an agent returns to its point of origin, or possibly some intermediate point, the data can be retrieved and verified in order to detect any modifications. With this method, agents can ask each host to sign new data and to append this (possibly encrypted) information to the agent before the agent moves to another host. If performed correctly, a malicious host will not be able to modify or view the signed data "because it cannot forge the signatures of other hosts" without access to private keying material [28]. This method is very useful in preventing potentially malicious hosts, which always have total access to a mobile agent's state, "from corrupting any partial results computed while the agent was at a previous host" [32, 42]. The downside of this method is that an agent that encrypts its data for privacy will not be able to use that data in any decision making until it returns to its original host. This is not necessarily bad, because many applications do not require use of intermediate data in order for their agents to function properly. Another disadvantage is that the size of the agent's data grows linearly with the number of hosts in an agent's itinerary.

A different method of encapsulating partial results is for an agent, just prior to migrating to another host, to encapsulate the results generated on the current host and ship them back in a message to the agent originator [6]. In this stateless scheme, data is never kept with the agent itself, but is always kept at the owner's site. The primary benefit of this approach is the immediate acquisition of knowledge of all intermediate

operations by an agent. The originator would not have to wait for an agent to return from its completed route before learning about its results. There are two primary disadvantages to this approach, however. First, this scheme greatly increases the amount of communication and bandwidth used on the network. Second, this scheme destroys the basic idea of autonomous agents – they should require little-to-no interaction with the agent owner during their operation. In addition, this method would preclude the possibility of an agent going off-line [23]. This scheme basically degrades to a form of client/server and thus loses the advantages of using mobile agents in the first place.

i. Itinerary Protection

Several approaches have been suggested for dealing with mobile agent itineraries [5, 41, 43]. One approach constructs an ordered sequence of sites to visit and loads this itinerary into the agent. "The agent uses this list to sequentially move from site to site" [44] through the network until it reaches its destination, usually the same as its origin. In this approach, the agent originator is responsible for determining the circuit for an agent before it begins migrating around the system [32].

In another approach, the itinerary is determined as the agent migrates, based on decisions it makes on its journey. While this approach allows more flexibility in agent movement, it is harder to protect these agents from being hijacked. No feasible scheme currently exists for protecting itineraries in this form.

A third approach maintains a "secure, verifiable audit trail which records the actual path followed by the agent" [41]. All major operations and sites visited are recorded during migration and encryption mechanisms are applied to protect the logs from

tampering [45]. When an agent reaches its final destination, the audit trail is examined to see where the agent has been. Any discrepancies or indications of tampering would then result in further action by the agent owner.

j. Clone Detection

An agent server can easily make copies of mobile agents that migrate to its site. A malicious server might do so in order to cause confusion for the agent system, or a host might inadvertently inject a copy of an agent into the network because of some faulty behavior. Multiple cloned agents would make it impossible to authenticate the original from the clone and might lead to multiple executions of the same agent [46]. Thus, the ability to detect agent clones would be beneficial to maintaining the security of a mobile agent system.

Baek [47] proposed one method of detecting agent clones and identifying which agent server generated them. In his method, a centralized coordinator executes a complex and detailed clone detection algorithm. Agent servers send this coordinator messages regarding an agent that it is currently hosting. The coordinator determines if any clones are present by examining the messages from all the agent servers. The agent server with a legitimate mobile agent receives a message back from the coordinator indicating that it can execute the agent. All clone agents receive messages that stop their execution.

This scheme suffers from several problems, the most obvious being that all agent interactions, to include agent creation, execution, movement, and destruction, must be pre-coordinated through the coordinator. This could be a major bottleneck in the system,

as well as a single point of failure. Likewise, the scheme assumes that the agent coordinator is always trustworthy and that all communications are secure [47], which is probably not a good assumption to make in a real-world network.

k. Fault-tolerance

Schneider [48] presents a fault-tolerant approach that attempts to deal with the effects of faulty hosts by replicating the agent at each visited site. The agent visits multiple hosts and each host sends the agent's output to every subsequent host to be visited where voting occurs to mask the effect of faulty hosts [37]. This approach uses significant bandwidth since every host must be notified of an agent's actions. In addition, this method cannot detect which hosts are faulty, only that a fault occurred somewhere in the network.

Another fault-tolerant approach has been proposed which "dispatches two identical agents to a known set of host systems, where one traverses the systems in one direction and the other agent does the reverse" [32]. The agent owner then compares the results of each agent when it returns to its originating host. Based on the comparison, the agent owner can pinpoint if a malicious host made an incorrect entry into an agent's data store. This approach can waste bandwidth and "the author of this approach acknowledged that this approach is only successful in detecting at most one malicious system" [37].

Another approach is to flood a system with agents at different entry points into the network and hope that one makes it successfully. This is a waste of bandwidth and offers no guarantees of working [37].

2. Example Mobile Agent Systems

With both industry and academia sponsoring active research in the area of mobile agents, many systems based on this paradigm have been developed over the past decade. These systems were designed to be integrated into any application domain that might benefit from using mobile agents. The subsequent sections give background information on the most popular and interesting mobile agent-based systems developed to date.

a. Telescript

Telescript, developed by General Magic in the early 1990s, was the first system designed for building distributed applications using fully mobile agents [49]. The intent was to use Telescript to build an "electronic marketplace" [49] in which producers and consumers of many types of goods and services could do business remotely over the Internet. Although the system did not succeed commercially, it became the system from which many other mobile agent-based architectures were subsequently developed.

The system had three major components: a specialized programming language with which to program agents, interpreters running on each host that could interpret this language, and communications protocols that allowed agents to move between hosts. The Telescript language was object-oriented and was portable among hosts with Telescript engines. The major communications components of the system were implemented in this language while application portions of each agent could be written in C or C++. Interpreter engines were located on each host that allowed agents written in this specialized language to be executed. These engines also implemented protocols

that enabled the packaging and transporting of agents to other hosts using high-level instructions by the programmer.

Telescript had limited security mechanisms. The only security features to protect agents themselves were very limited forms of authentication and encryption used only during agent transfer to prevent eavesdropping [50]. Since most of Telescript's security features were discretionary, there was always a risk of a programmer failing to correctly use them. Also, since Telescript assumed that all machines were trustworthy [51], agents did not have to be authenticated on each move [50], providing opportunities for spoofing, Trojan horses, and other attacks.

Even though Telescript was the first commercial mobile agent system, it failed to gain market share because the proprietary language was difficult to learn and the network architecture was difficult to integrate. General Magic subsequently re-implemented its original ideas on mobile agents in a Java-based product called Odyssey. Developers can use Odyssey's Java class library to create mobile agent applications for their particular domain. The security concerns for Telescript, however, also remain for Odyssey.

b. D'Agents (Agent Tcl)

D'Agents, formerly known as Agent Tcl, is a mobile agent system developed at Dartmouth College in which agents are written using Tcl scripts, Java, or Scheme [51, 52]. The primary component of a D'Agents system is an agent server that runs on each host that supports an interpreted execution environment for mobile agents as they travel through the network. D'Agents also implements a high-level inter-agent

communication protocol similar to RPC that enables agents to pass messages to each other through a procedure call abstraction or by setting up a streaming connection [52].

D'Agents security focus is on protecting machines from malicious agents, not protecting agents from malicious machines [52]. While D'Agents provides no built-in mechanisms for protecting agents, it does call an external cryptographic tool, Pretty Good Privacy (PGP), to perform authentication checks and for encrypting data in transit. It is not required, however, that an agent's identity be re-verified at each stop [53] because D'Agents servers are typically configured so that all hosts under a single administrative domain trust each other [51]. Gray et al. also detail several other weaknesses of the system's authentication scheme, including serious vulnerabilities to replay attacks [51]. In addition to these shortcomings, no means exist of ensuring availability and fault tolerance of agents should they become lost or intentionally destroyed. If a host goes down while an agent is executing, the agent is lost [52].

c. Mole

Mole, developed in 1994 at Stuttgart University in Germany, was the first mobile agent system that was developed using the Java programming language [54]. In Mole, agents are active entities, which move from place to place in order to rendezvous with other agents and to access various host services. Mole implements agents as multi-threaded objects where both agent code and data are transferred during migration. Mole also addresses agent-to-agent communication by supporting both message passing and remote method invocations (RMI).

Mole addresses the protection of its agents by implementing a 'blackbox' [34] out of an agent using algorithms to obfuscate the agent's code, making it hard to understand [54]. These techniques offer no guarantee of agent protection because the degree of security is not quantifiable. The authors of Mole identify other mechanisms that could be used to afford better protection for their agents, but none of these techniques are implemented.

d. Ara (Agents for Remote Action)

Ara [55], developed at the University of Kaiserslautern, is a multi-language system that has recently added support for Java-based mobile agents. Ara depends on an altered implementation of the Java Virtual Machine (JVM) running on each of its UNIX platforms. By altering the JVM, Ara supports transparent continuation of an agent at any arbitrary point of execution after a migration, thus fully preserving agent state. Ara also implements a form of agent communication based on the client/server style of interaction, but leaves the actual choice of communication language open to the developer.

Ara addresses some of the basic issues of agent protection using well-known public key cryptographic primitives. Agents can be optionally encrypted during migration to protect them from eavesdropping. In addition, the Ara developers suggest the possibility of future security measures to provide authentication, but none of these mechanisms are currently implemented.

e. Aglets

Aglets are mobile agents developed by IBM running on a Java-based system [56]. Like other mobile object systems, agents migrate between agent servers located on each host in the network. Since the agents are written in Java, thread-level execution state cannot be saved between migrations. Therefore, Aglets use an event-based scheme in which specific methods are invoked when certain events occur in its life cycle. For example, before an agent leaves a server for another host, the *onDispatching()* method will be called which handles all the details of removing the agent from its current context and inserting it into its destination context [56].

While providing interesting solutions to protecting hosts from malicious agents, Aglets currently have very limited support for the security of agents against malicious hosts. Karjoth et al. briefly describe possible attacks on its system, including eavesdropping, replay, masquerade, and repudiation attacks [56]. Their current implementation, however, does not address these agent protection issues [56], instead leaving room to possibly build these mechanisms in future versions of the system.

f. TACOMA

The TACOMA project [57], another mobile agent system from the early 1990s, focused primarily on operating system support for agents. Agents are implemented using Tcl and each site on the network runs a Tcl interpreter on which agents may execute. Because agents are written with scripts, thread-level state cannot be saved when an agent migrates; therefore, all agents restart their programs when they arrive at another host.

TACOMA almost completely ignored the need for security, so no security mechanisms were implemented. It did, however, provide hooks so that agent authors could add their own cryptographic subsystems [51]. The TACOMA team also included a rudimentary form of fault tolerance for its agents, using a combination of checkpointing and rear-guards for tracking mobile agents as they migrated [57]. This provided the limited ability to generate a new agent in cases where a failure caused a previous agent to disappear from the system.

g. Concordia

Concordia is an agent-based framework that supports developing and executing mobile agents written in Java [53, 58]. An Agent Manager on each host provides the execution environment for agents as well as the basic mobility mechanisms necessary to send and receive agents. Concordia provides inter-agent communication based on either distributed events or agent collaboration [53].

While Concordia provides many useful mechanisms for supporting the mobility of its agents and inter-agent communication, the security support is limited. Concordia does protect its agents in transit by encrypting and authenticating them using Secure Sockets Layer (SSL) encryption [59, 60]. The system also protects persistent agents that are waiting on disk through a variety of encryption techniques. Concordia assumes that its servers are trusted, so once an agent reaches a Concordia server, no protections are in place to protect an agent while present in memory from a malicious host [53]. Concordia provides a reliable queuing mechanism to ensure reliable agent mobility between host pairs, however this mechanism does not ensure global availability of

agents as they traverse the network [53]. For example, if an agent disappears while executing on a host, no system-wide mechanisms exist for regenerating the agent.

h. Ajanta

Ajanta is a Java-based system developed at the University of Minnesota in which Java's serialization features are used for agent state capture before migration [5]. Ajanta contains the basic mechanisms any mobile agent system needs, but differs somewhat from other systems in the structure of its agents. The code an agent may need is not carried along with the agent, but rather code is loaded on demand from an agent-specified server [41]. Communication between agents is achieved using Java's Remote Method Invocation (RMI) facilities.

Ajanta provides limited mechanisms for protecting privacy and authenticating agents during migration, but, does not address the issues of agent availability, regeneration in case of loss, or the prevention of denial-of-service attacks against agents [41].

3. Mobile Agent Communications

Mobile agents may need to interact with other components in an agent-based system in order to successfully complete their tasks. Agents must have some mechanism in which to interact with their local execution environment, with other agents, and with other application components in the network [61]. The models of agent-based coordination can be classified based on temporal and spatial coupling [62]. In spatial coupling, agents and other system components "must share a common name space in order to communicate" [62]. That is, an entity must know the name of the other entity to

which it wishes to communicate. In temporal coupling, communication is based on whether the sender and receiver of any communication must both exist at the time the actual communication takes place [62]. Temporally coupled communication models are, by definition, synchronous. This basic taxonomy of mobile agent communication is shown in Figure 5.

Direct communication is both spatially and temporally coupled. Entities send messages directly to one another. Recipients of any communication must exist prior to the communication and senders must know the name of the receiver. While most current client-server systems use this model for communications, it is not well-suited for mobile

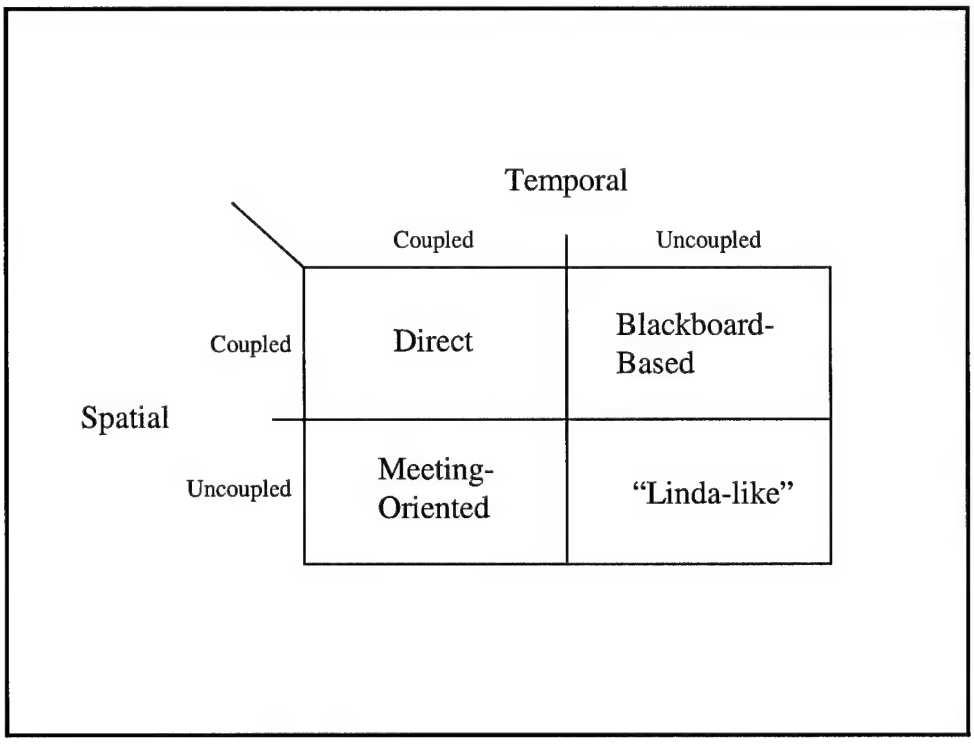

Figure 5: Mobile agent communication taxonomy [62].

agent communication for two reasons. First, since components are mobile, a message sender may not know the location of the receiver at the time of communication. In addition, if agents are created dynamically, it may be difficult to determine which agents exist at any given time [62].

Meeting-oriented communication models are temporally coupled, but spatially uncoupled. An agent does not necessarily have to know the name of the intended recipient of it communication. Instead, agents join at predetermined meeting points where they communicate with other entities in the system. However, all parties must pre-exist for any synchronization and communication to take place [62].

Blackboard-based models are temporally uncoupled, but spatially coupled. These systems rely on a repository to store and retrieve messages. In this model, the sender leaves a message in the blackboard where the receiver can retrieve it at some subsequent time [62]. Because messages are left in the blackboard, there is no need for the sender and the receiver to exist at the time of the communication. This model is well-suited for "a highly dynamic environment, such as the Internet, where it is very difficult to know the presence and the location of a given agent" [62]. This is the model that was chosen for this research, for reasons indicated in Chapter III.

The fourth coordination model is named "Linda-like" by the authors of the taxonomy after the Linda parallel programming language [62]. This model is fully uncoupled, both spatially and temporally. Similarly to the blackboard-based model, a repository exists for the storage and retrieval of messages. In this model, however, the messages are not addressed to a particular recipient. Instead, entities retrieve messages using Linda-like

tuples in which a pattern matching mechanism permits an agent "to retrieve a message by specifying only few parts of the message itself" [62].

C. Intrusion Detection Systems

The goal of any intrusion detection system is to identify that an intrusion has been attempted, is currently underway, or has occurred in the past [63]. This usually involves some form of dynamic monitoring of the actions that are taken in a system and deciding whether those actions are indicative of an attack [64]. In order to perform this function, intrusion detection systems need some knowledge-base of known attacks, information on the current state of the system being monitored, and audit information that describes the events that have occurred in that system [64]. Based on this information, two significant questions arise: "what data should be collected and where should the analysis of this data be accomplished" [65]. Different systems answer these questions in different ways.

Intrusion detection systems, however, are not a cure-all for computer security. As systems grow and become more connected, the straightforward task of even noticing a penetration or penetration attempt becomes increasingly difficult. Often times, intrusion detection systems cannot distinguish between events that can be considered normal and those events that indicate intrusive behavior [66]. These systems are vulnerable to mistakenly characterizing normal behaviors as attacks, to missing attacks altogether, to being disabled, and to incomplete or incorrect knowledge about what should be considered an attack [67]. The sheer quantity of information itself needed to monitor a network frustrates an intrusion detection system's ability to rapidly identify problems. In

addition, like many other security mechanisms, intrusion detection systems have a very difficult time dealing with the insider threat.

1. Agent-Based Intrusion Detection Systems

Much of the research currently ongoing in the field of mobile agent-based intrusion detection systems focuses on either mobile agents or intrusion detection, with little work on integrating these two into one seamless system. To be successful, both areas need to be examined together to identify issues that can only be addressed in an integrated system. While the types and categories for intrusion detection systems are many and varied, this section describes the previous research that has been done in the area of host-based intrusion detection systems that have some agent components (whether mobile or not). Currently, only a few systems have been proposed that approximate this approach. A survey of these systems follows.

a. AAFID (Autonomous Agents for Intrusion Detection)

Purdue University has proposed an architecture called AAFID (Autonomous Agents for Intrusion Detection) for building intrusion detection systems that use agents organized in a hierarchical fashion for data collection and analysis [68]. The system architecture is built around three primary components: agents, transceivers, and monitors.

An agent is an entity that constantly monitors activity for a specific host on which it resides and "reports abnormal or interesting behavior" to a higher-level entity, a transceiver [68]. For example, an agent could be looking for a large number of login

failures. If some threshold is reached, the agent would report the activity to its corresponding transceiver for action. Agents themselves cannot generate an alarm, but instead rely on a transceiver or monitor to generate an alarm based on their more complete knowledge of the system. As transceivers combine the different reports from the various agents on the host that report to it, it builds a picture of host-level activity for the specific machine it is monitoring [68]. All communication between AAFID agents is done through a transceiver, which filters the information and decides what to do with it based on how the agents on the host are configured [68]. Agents can have any functionality the system developers desire – some can be written as simple monitors for certain activities and others can be more complex entities that do analysis on a set of observations. While agents can be updated or added to the system, agents are not mobile once they have been dispatched to a host transceiver [1].

Transceivers are located on each host in the network and provide oversight of all agents running on its host. They give local commands to agents on their host and also perform data reduction on the information provided by the agents. Since transceivers reside on all hosts that need to be monitored in the network, they can serve as external interfaces between hosts in the system. Transceivers respond to commands issued by higher-level monitors and also provide these monitors with information provided by agents about host activity as necessary.

Monitors are high-level entities that have access to network-wide data to provide system-wide command and control of the other entities. The functionality of monitors and transceivers is similar. The main difference between the two entities is that monitors

control entities residing on several different hosts, whereas transceivers control only agents residing on a single host [68]. Monitors have the ability to detect events that might not be detected by agents or transceivers, such as suspicious behavior among several hosts. Monitors can themselves be organized hierarchically, to reduce the amount of data to be processed and to provide for increased scalability. Higher-level monitors communicate with the user interface to provide feedback on the system and to obtain command and control information from the user.

While acknowledging that security should be provided for its stationary agents, no mechanisms are in place to provide for agent protection. Also, the system developers understand the need for security in the communications components of the system, but no encryption or authentication mechanisms are currently implemented, making the system vulnerable to many types of attack. Adding security features to this system has been noted as future work.

b. Hummingbird

The University of Idaho has developed a prototype IDS called the Hummingbird System [69]. The system attempts to address many of the issues involved in detecting intrusions on networks whose sites span across domains that are not under a central authority. Hummingbird is a prototype distributed system built for managing misuse data to support this form of cooperative intrusion detection across multiple domains [69, 70].

In Hummingbird, an agent is assigned to a host or set of hosts and communicates with other such agents in a peer relationship located on different machines in the network. A

group of manager entities also exist which are responsible for transmitting commands to other managers and to their subordinates. Agent communication is not restricted to entities just within its local domain, but can also communicate with entities on hosts that reside in other domains. Agents continuously collect data on host activity and perform simple filtering for both data reduction and data sanitization. Individual agents make decisions on whether to share data they have collected based on their own locally controlled policies on trust, information flow, and cooperation between entities [69]. All communication between agents is encrypted and authenticated using the Kerberos system [71]. Hummingbird does not provide autonomy nor mobility to its agents, nor does it address issues of agent availability or prevention of denial-of-service attacks. Hummingbird does, however, offer some unique mechanisms for distributed data sharing across administrative domains that might be useful in a mobile agent-based intrusion detection system.

c. Intelligent Agents for Intrusion Detection

A project at Iowa State University is based on intelligent agents for intrusion detection [72]. It proposes an artificially-intelligent design and architecture for an IDS built from distributed components that use data mining techniques to monitor system activity. This system offers no mechanisms to quickly tailor, specialize, or change its agents once they are in place. In addition, it would not be fielded as a real system because of its lack of security for its components. Security is not addressed in their research and no form of agent protection is provided. It appears that all of the

components of this system are vulnerable to attack. Adding protection mechanisms would greatly enhance the viability of using this system in a real-world setting.

d. Intrusion Detection Agent System

The Information-technology Protection Agency (IPA) in Japan is working on a project called the Intrusion Detection Agent system (IDA) [73]. IDA has two characteristics that are uncommon in most conventional IDSs. First, it uses mobile agent mechanisms for tracing the origin of intrusions among various hosts in the system. Second, its mechanisms for watching for suspicious behavior are based on watching for specific events that relate to possible intrusions, rather than analyzing all user activities. If a suspicious event is detected, mobile agents collect information related to the event in order to determine its origin.

IDA deploys sensors on each host to monitor system logs in search of activity that may be indicative of an intrusion. If such activity is found, the sensors send information to a manager which is responsible for determining if an intrusion has actually occurred. If a manager makes a positive determination, it will dispatch a series of agents in an attempt to gather more information on the suspected intrusion and to trace the intrusion through the network.

IDA employs two types of agents: one to trace the path of a suspected intrusion back to its point of origin and one to collect information on possible intrusions. When a sensor reports an indicator of possible intrusion to a manager, the manager first dispatches a mobile tracing agent to autonomously trace the intrusion from host to host back towards its point of origin. As it migrates, the tracing agent activates an

information-gathering agent for each host it visits. Instead of each host sending its logging information back to a central server, and thus using massive amounts of bandwidth, these information-gathering agents collect information related only to the suspected intrusion and then move to a manager where its data is analyzed. As all of the tracing and information-gathering agents collect information, they send it to a common bulletin board where all system entities can get a system-wide view of what is happening. The manager would then make a determination whether an intrusion actually occurred by evaluating information in the bulletin board [73].

IDA is being implemented using D'Agents, which means its agents have all the vulnerabilities previously mentioned for this mobile agent system. While the use of mobile agents in this IDS are novel and interesting, IDA would benefit greatly if security mechanisms were added to protect its agents.

e. EMERALD (Event Monitoring Enabling Responses to Anomalous Live Disturbances)

EMERALD is a distributed intrusion detection system developed by SRI International that uses "distributed, independently tunable, surveillance and response monitors" as its primary component [74]. These monitors, similar to agents, analyze and respond to malicious attacks on local hosts. They also support the EMERALD framework for coordinating host analyses enabling the system to better detect network-wide attacks.

While EMERALD's service monitors are dynamically deployable, they remain stationary once they are dispatched. Thus, these components cannot be considered

mobile agents. EMERALD does, however offer some novel approaches to network-wide surveillance that can be applied to other intrusion detection systems which are based on mobile agent architectures.

The only security requirements included in EMERALD involve its messaging system. Messages between monitors and other components in the system can be protected using pluggable transport modules [74] to ensure communication security, integrity, and reliability.

D. Cryptography

The cryptographic algorithms underlying many of the protection mechanisms used in this system have been well researched for many years. The issues of providing confidentiality, integrity and authenticity for mobile agents can be addressed by using a combination of these techniques.

1. Encryption

A number of cryptographic algorithms can be used in order to provide confidentiality for information that must navigate an insecure network. All modern algorithms use a mathematical function for both encryption and decryption, in which the function relies on some key or set of keys in its operation. These key-based algorithms are classified as either symmetric or asymmetric based on how the keys are used to encrypt and decrypt information.

In symmetric key algorithms, the key used to encrypt data is the same key used to decrypt data. In cryptosystems based on symmetric algorithms, the sender and receiver

must agree on a key prior to any secure communication. The key must remain secret from any third party as long as the communication must remain secret. Figure 6 illustrates this process.

Symmetric key algorithms can be classified into two categories based on how the algorithm processes data to be encrypted. Stream ciphers encrypt data one bit (or byte) at a time. Stream ciphers are often implemented in hardware because they are efficient in silicon. Such hardware implementations are often found in digital communications equipment because bits can be encrypted as they go by [75] and they have no error propagation [76]. Block ciphers encrypt data in groups of bits called blocks. Block ciphers are often implemented in software because they avoid time-consuming bit operations and they more easily operate in computer-sized blocks of data [75].

Asymmetric algorithms, commonly referred to as public-key algorithms, use a pair of keys: one key is used for encryption and another is used for decryption [77]. These algorithms are based on the theory that the decryption key cannot be calculated from the encryption key in any reasonable amount of time. The encryption key can be made public and thus is called a public key. The decryption key is kept private and is only known by the owner. Thus, anyone can send a person a secret message by using the receiver's public key to encrypt the message. Only the receiver with the corresponding private key will be able to decrypt and read the message. Figure 6 illustrates a typical public key algorithm.

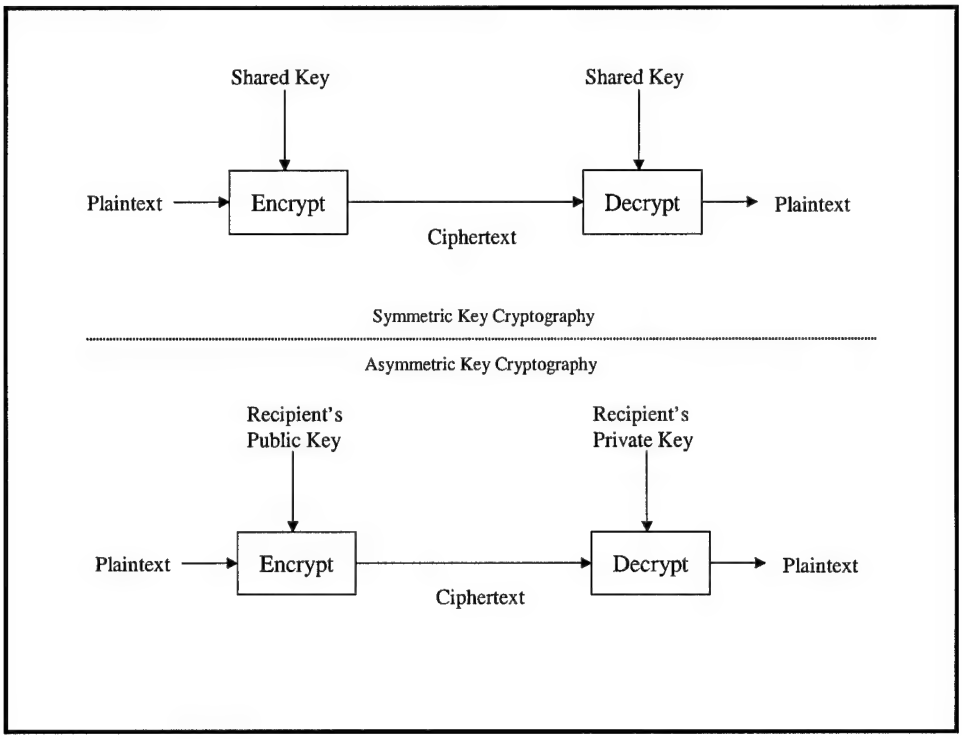

Figure 6: Symmetric and asymmetric cryptography.

2. Digital Signatures, Integrity, and Authentication

Public key algorithms can also be used to produce digital signatures. If a message is encrypted with the sender's private key, then the recipient of the message can ensure that the message came from the correct person because only the sender's public key decrypts it. Using a public key algorithm in this manner provides an excellent means of authenticating messages.

A cryptographic primitive central to public key cryptography is the one-way function. One-way functions have a unique property: they are easy to compute, but are

significantly difficult to reverse. In other words, given some value x, it is easy to compute $f(x)$, but given $f(x)$, it is computationally infeasible to compute x [75, 76].

A special form of a one-way function is the one-way hash function, also commonly referred to as a message digest, cryptographic checksum, or message integrity check. A simple hash function takes a variable length input, sometimes referred to as the pre-image, and converts it to a fixed-length output called the hash value [75]. A one-way hash function is a hash function that only works in one direction; that is, it is easy to compute the hash value from the input string, but computationally difficult to generate an input string that hashes to a particular hash value. A good one-way hash function that is suitable for use in cryptographic protocols should be collision resistant, meaning that it is computationally infeasible to generate two different input strings which hash to the same output [76].

A commonly used one-way hash function algorithm is the Secure Hash Algorithm (SHA-1). The SHA-1 takes a message of any length less than 2^{64} bits as input, and produces a 160-bit message digest as output [78]. The SHA-1 is considered secure because it is computationally infeasible to recover the message corresponding to a given message digest, or to find two different messages which produce the same digest [75]. The SHA-1 is effective because any change to a message will result in a significantly different corresponding message digest causing the verification of the signature to fail. The SHA-1 is based on a similar algorithm for generating message digests, MD4 [76], but provides increased security against brute-force attacks since it produces a 160-bit digest compared to MD4's 128-bit output.

A variant of the one-way hash function is the message authentication code (MAC), also known as a cryptographic checksum. A MAC is simply a one-way hash function with the addition of a secret key [75]. The purpose of a MAC is to provide assurances about both the source of a message and its integrity [76]. The MAC value is a function of the input string and a secret key. Only persons with knowledge of the secret key can verify the hash value.

This technique assumes that two parties that want to communicate, say X and Y, share knowledge of a secret key S_k. If X wants to send a message to Y, it will calculate the MAC as a function of the message M and the secret key. The message and the MAC are then transmitted to Y. Recipient Y then performs the same calculation on the message that was received and compares the received MAC to the newly calculated one. If only X and Y have knowledge of the secret key, then the receiver is assured that the message was not altered and that the message was indeed sent by the alleged sender [78].

CHAPTER III

DESIGN

A. Proposed Methodology

The overall intent of this research was to develop a methodology for protecting mobile agents in intrusion detection systems and to demonstrate the ability of such agents to address the shortcomings in current host-based IDSs. This methodology supports the defense of computer systems through a secure, mobile agent-based architecture.

The system design to support the proposed methodology for protecting the confidentiality, integrity, and availability of mobile agents is summarized in Figure 7. This design involves the interaction of a number of distributed components and tools. The components and their interactions were designed to provide security for each aspect of the system. Within each component, a number of functions are outlined. Each of these various components, their functions, and their interactions are discussed in the following sections.

1. System Overview

Security is of fundamental importance in this system. In order to build an attack-resistant architecture in which mobile agents can execute and migrate, each of the components in the system were designed from the beginning with security in mind. Although introducing strong security mechanisms appear to conflict with flexibility and

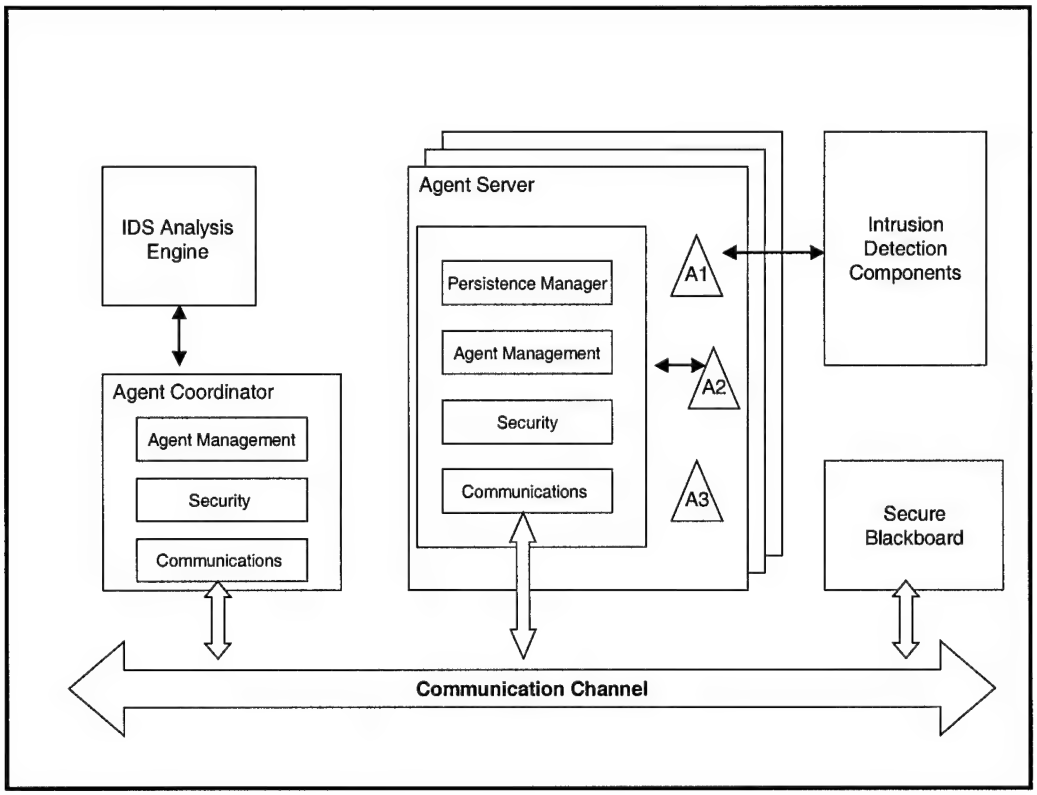

Figure 7: Proposed methodology.

usability, not addressing security issues prevents such mobile agent based systems from being fielded as real-world applications [79]. The interactions between components and the security implications of each is described next for each element of the system.

2. Mobile Agents

A mobile agent is simply "a program that represents a user in a computer network and can migrate autonomously from node to node, to perform some task on behalf of the user" [5]. Once a mobile agent is dispatched, it becomes independent of the creating host and can operate asynchronously and autonomously. A fundamental characteristic

of mobile agents is that they should only reside on one host at a time; however, since mobile agents move from host to host, they can occupy different hosts at different times. In addition, agents should be independent of one another in that their executions can be performed concurrently on different hosts [16].

Mobile agents in this system are composed of three basic components: interpreted code, data (or payload), and an itinerary as illustrated in Figure 8. Agents were designed to either follow a predetermined itinerary that is established by the agent coordinator (discussed below), or to follow a random path based on user established criteria. Upon arrival at a host, the agent server first checks the incoming agent's security credentials before executing the agent's code. Data that results from an agent's execution on a particular host is stored in a reserved location as payload for that host. Upon the return of an agent to the agent coordinator, analysis is performed on the collected data to determine if any indications of security problems are present.

This design used agents in a roaming paradigm. That is, many different agents, with different functions, are dispatched to roam through the network looking for potentially intrusive behavior. These agents collect information specific to their particular purpose as they travel. Agents can be designed very generally to just gather data as they move from host to host. Others may be designed to look specifically for particular evidences of security problems. Some agents may be written to deliver information to hosts as they travel (i.e. delivering new attack signatures to stationary host-based IDS components). Using an adequate number of functionally disparate agents at a reasonable frequency will provide sufficient network coverage to detect many potentially serious

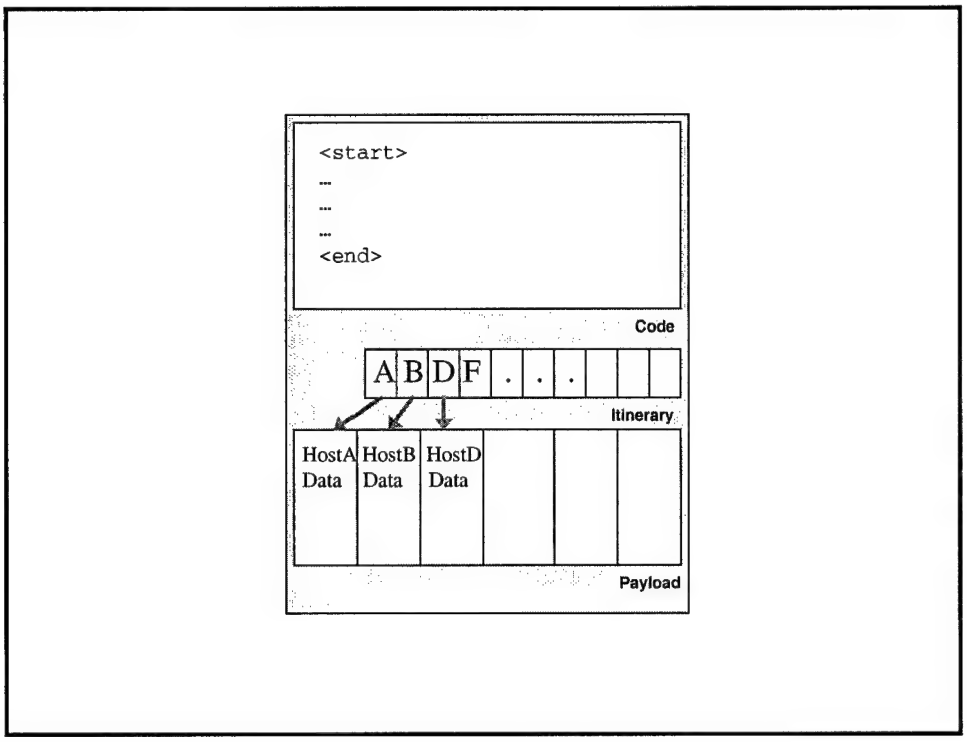

Figure 8: Basic structure of a mobile agent.

system vulnerabilities and intrusive behaviors. Regardless of the type of agent, however, an agent's own security must be ensured in order for this approach to be viable.

Mobile agents must be protected from three major threats. First, the confidentiality of an agent's data and code must be protected. An intruder may gain valuable information to help him better penetrate a system if he is able to see which mobile agents are migrating through the network. In addition, if intruders can see data that has been collected by agents at previous hosts, they might gain an advantage in further exploiting the system. The confidentiality of agents, therefore, must be protected from eavesdroppers in the network.

Another major threat for mobile agents is tampering. The integrity of mobile agent code and data must be protected. Any changes to an agent must be detected as quickly as possible. Agent servers must not execute code that may have been tampered with in order to prevent malicious changes from doing any harm. In addition, tampered data must not be relied upon as a true representation of the state of the system, in order to prevent possible intruders from hiding their tracks through agent manipulation.

Finally, the availability of agents must be protected by ensuring the system is resilient to denial-of-service attacks and faulty hosts [80]. Mobile agents may be inadvertently removed from the system by a faulty host or they may be intentionally deleted by a malicious host or intruder. In either case, the system must be able to detect if an agent is removed and regenerate lost agents back into the system. The system architecture was designed to address and counter each of these threats.

In order to achieve the goal of providing confidentiality for mobile agents during migration, each agent is encrypted before it is transported between hosts. Public key cryptography offers many ease-of-use benefits over symmetric-key encryption, especially when supplemented with a public key infrastructure. Besides offering confidentiality of data, public key encryption also forms the basis of providing mutual authentication of hosts and agents. All of the components in the designed architecture were designed to use public-key cryptography for both secrecy and authentication as described in detail in Chapter IV.

In addition to protecting an agent during transmission, agent confidentiality must also be protected while agents are resident on agent servers. Because agents must exist in

plaintext form in order to be executed, they may be vulnerable to eavesdroppers on the host while executing. No general solution has been found to protect agent code that is executing; however, Sander and Tschudin [39] proposed a very limited method of code protection using encrypted functions. The major drawback of this approach, though, is that no known methods exist in which arbitrary functions can be transformed into encrypted forms for use in mobile agents.

Agent data can still be protected, however, using conventional cryptographic mechanisms. Agents in this system were designed with a secure payload repository that keeps data private during the course of an agent's travels. Each host places data relevant to its site in the repository and encrypts that data using public-key cryptography. This ensures that both the confidentiality and integrity of the payload is protected both during transmission and while an agent is executing.

In addition to protecting the privacy of agent code and data from eavesdroppers, an agent's integrity must also be protected. Any changes to an agent's code, data, or itinerary must be detected as soon as possible before any damage can be done. Digital signatures are used to detect any tampering of an agent. Public-key cryptography and SHA-1 generated message digests provide a convenient mechanism for ensuring the integrity of mobile agents. Agent servers were designed to sign an agent prior to migration. Upon arrival at its destination, the receiving agent server checks the agent's digital signature to verify that there was no tampering of the agent during transit.

In addition to signing the entire agent, each of the components that makeup an agent are also digitally signed as illustrated in Figure 9. The Agent Coordinator signs an

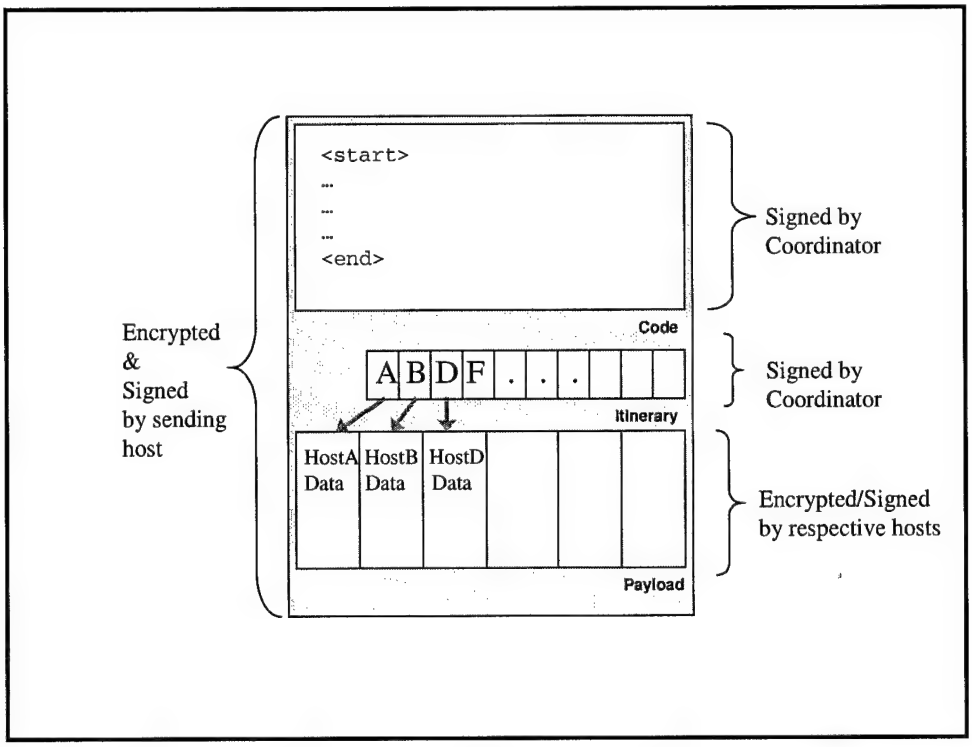

Figure 9: Protection for a mobile agent.

agent's code and itinerary before dispatching it. This ensures that any tampering of these components can be detected upon arrival at a subsequent host. Each host also signs the data it adds to the payload so that any changes made subsequently will be detected upon the agent's return to the agent coordinator.

Mobile agents and agent servers do not necessarily trust one another. This issue of mutual suspicion was addressed in the system by ensuring that mechanisms for mutual authentication were in place. Each time an agent migrates, it authenticates the host on which it arrives and the receiving agent server authenticates the agent to verify its origin. Public key cryptography is used as the authentication mechanism. In addition, each

agent is assigned a unique and unchangeable agent identifier when it is created, which helps prevent replay and flooding attacks.

Mobile agents are associated with a security level that can be changed by the Agent Coordinator prior to dispatch. The security level can be configured by the system administrator to provide flexibility and an appropriate level of security for the environment in which agents exist. Level 1 agents are the most secure, in that constant communication with the coordinator is maintained at each hop. Level 2 agents are equipped with medium security, in that faulty hosts are circumvented (taken care of by each agent server) and only asynchronous messaging occurs with the coordinator when needed. Level 3 is low security, in that no communication occurs with the coordinator between the dispatch of an agent and its eventual return. Care is taken to recognize when an agent never returns. In addition, agents may followed randomized itinerary routes if the system administrator so chooses. The role of these security levels will be described more in subsequent sections.

3. Mobile Agent Servers

Mobile agent servers provide the actual runtime environment for visiting mobile agents. Because most networks are composed of heterogeneous components, the runtime environment provided by agent servers supports an interpreted programming language [81, 82]. Each agent server must have a unique name and must be running on each host in the network that will support mobile agent execution. In addition to providing the runtime environment for mobile agents, these servers also handle the local

management, communications, transport, and regeneration of mobile agents. These functions are handled by subcomponents of each agent server.

The agent management component for an agent server handles all management functions for the agents currently resident. The agent management functionality serves as the director of all agent activities for that particular host. It also keeps track of the agents that are running on the server and answers any queries from the Agent Coordinator about their status.

The agent transport component supports the migration of agents to and from the host. This component is responsible for serializing agents prior to their departure and deserializing them upon their arrival. In addition, the agent transport component works with the security components to encrypt and decrypt agents, check the code and itinerary integrity of arriving agents, and authenticate the origin of incoming agents. The agent transport component is also responsible for ensuring the successful migration to the next host on an agent's itinerary. Any problems with agent transport are handed off to the agent management function which then makes decisions on how to address any errors.

The communications component is responsible for all remote interactions between the agent server and other system components. This includes communications with the other agent servers, the secure blackboard, and the agent coordinator. All communication takes place using asynchronous messaging via socket connections.

Four types of duplex communication exists in the system:

- Agents to Agent Server.
- Agent Coordinator to Agent Server.

- Agent Coordinator to Agent (through Secure Blackboard).
- Agent to Agent (through Secure Blackboard) – one of the primary motivations for mobile agents is to avoid remote communications in the first place [16], so communication between agents takes place asynchronously through a Secure Blackboard (secure shared memory).

The persistence manager is responsible for storing mobile agents to nonvolatile storage to aid in the recovery and regeneration of agents that may become lost during migration to other hosts. "The notion of a partial failure is a fundamental characteristic of a distributed system: at a given time, some components of the system might have failed whereas others might be operational" [83]. This function is critical in a distributed system where defective transmissions, faulty hosts, and malicious components can exist.

An agent is automatically stored to disk before each migration as determined by the system's fault-tolerant policies described later. In the case of a subsequent fault or malicious deletion of an agent, the persistence manager may be required to regenerate a stored agent which can then be reintroduced into the network. Upon notification that an agent is no longer needed, the persistence manager removes stored agents from the persistence store to free space for other agents.

The interface between mobile agents and the intrusion detection components on the local machine takes place through proxy agents. These proxy agents are stationary, trusted components dedicated to representing the intrusion detection components to the agent server. The proxy agent oversees the security of all interactions between visiting mobile agents and the IDS components. These interactions take place through

application programming interfaces (APIs) that allow mobile agents access to the IDS system.

4. The Agent Coordinator

All mobile agents originate at the Agent Coordinator. The Agent Coordinator maintains information about all components in the mobile agent system, including hosts, agent servers, and agents. As hosts and agent servers are added to the network, they are registered with the Agent Coordinator. The Agent Coordinator is also responsible for the creation and removal of mobile agents from the system. The Agent Coordinator, in conjunction with individual agent servers, is responsible for detecting breaches in security of any of its agents, including violations of integrity and availability. Finally, the Agent Coordinator receives instructions from and sends data to the intrusion detection system analysis component which performs analyses on data collected by the mobile agent system.

The Agent Coordinator incorporates three major functions. The agent management function is responsible for system-wide management functions for all agents and agent servers in the network. The management component creates agents, assigns each a unique identifier, and determines the agent's itinerary. This component also maintains timers on each dispatched agent in order to detect when agents are lost or destroyed. Timers are also maintained in order to automatically dispatch agents without user intervention.

The communications component is responsible for all remote interactions between the agent coordinator and all other system components, including communications with

the agent servers, the secure blackboard, and the IDS Analysis Agent. All communication takes place using asynchronous messaging via socket connections.

The security component of the Agent Coordinator is responsible for detecting violations of agent integrity and for ensuring the availability of all dispatched agents. Upon return of an agent to the Agent Coordinator, the security component performs an analysis of the agent to determine of any security violations have occurred. This component checks the integrity of the agent's code, itinerary, and payload. If data for a host that was supposed to be visited on the itinerary is missing, the security component may perform further analysis to determine the cause as described in the section of distributed fault tolerance. The management of trust [84] in the system is addressed by restricting key management of all public keys to the agent coordinator and a public key infrastructure.

5. Intrusion Detection System Analysis Engine

Agents dump their payload to persistent data logs upon return to the Agent Coordinator. The IDS Analysis Engine is responsible for analyzing agent payloads for indications of intrusion or security violations. The IDS Analysis Engine maintains information on attacks and system vulnerabilities and compares agents' payloads against this information. If an analysis indicates a possible problem, the IDS Analysis Engine can then take some corresponding action, usually involving some form of notification to the system administrator.

The IDS Analysis Engine was designed to be modular, in order to be easily expandable when new types of analyses must be performed for new types of attacks or

vulnerabilities. This enables the system administrator to add additional functions that look at agent payloads for intrusive behaviors that may not have been previously defined. Similar to virus scanning engines, the IDS Analysis Agent may have to be updated by adding additional functionality as new types of vulnerabilities and attacks are defined that the previous modules were unable to detect.

The Agent Coordinator and the IDS Analysis Agent can communicate directly. Typically, this communication entails the Agent Coordinator informing the IDS Analysis Agent that new data has arrived that requires analysis, and the IDS Analysis Agent loading that data for investigation. The IDS Analysis Agent can also request that additional actions be taken by the Agent Coordinator, such as dispatching agents or querying agent servers in order to gather more information about a potential problem.

6. Secure Blackboard

Agents communicate asynchronously through the Secure Blackboard. Agents can check for messages on the Secure Blackboard when arriving and leaving hosts, or at any other time during their execution as determined by their authors. The Agent Coordinator and agent servers, likewise, can check for messages periodically during their execution. Upon receipt of a message, these entities will then act appropriately based on the type of message and its content.

The Blackboard is similar to an agent bulletin board system [85], but is shared by all hosts on the network. Components of the system send messages by sending them to the Blackboard, where they may be subsequently received at some later time. Since this communication protocol is temporally uncoupled, but spatially coupled, these systems

rely on a repository to store and retrieve messages. Because messages are left in the Blackboard, there is no need for the sender and the receiver to exist at the time of the communication. This model is well-suited for the highly dynamic environment that is required by this research.

7. User Interface

The user interface to the agent coordinator allows the system administrator to manage and oversee the entire system. Specifically, the user interface allows the user to easily monitor, create, dispatch, and delete agents from the system. In addition, the user interface notifies the administrator of potential security violations in the system, including violations of agent integrity or availability. The user interface was designed to be a simple window-based system, incorporating menus, icons, and dialog boxes to facilitate easy interaction with the user. Because the user interface is the gateway into the entire system, all users' actions are authenticated and logged by the system. The IDS Analysis Agent and the agent servers also provide graphical user interfaces for functional, debugging, and maintenance purposes.

B. Distributed Availability for Agents

In addition to addressing the issues of confidentiality, integrity, and mutual authentication, the issue of agent availability is important because of the threat of denial of service. Mechanisms must be in place to ensure that the system recognizes when agents are lost or destroyed, and that lost agents can be recovered. This section explains the design of the these mechanisms for the proposed methodology.

The two primary threats to agent availability are loss of agents (possibly through malicious intent) and host servers that malfunction. Agents may be lost because of failures in one of the communication layers of the network or because an agent server goes down while hosting an agent. Servers may malfunction for a number of reasons, including hardware problems, software problems, or intrusive activity. Regardless of the reasons, the system was designed to recognize when agents are lost and when servers malfunction.

The security level associated with an agent determines the protocol that is followed for detecting missing agents. Level 1 agents are the most secure, in that constant communication with the coordinator is maintained at each hop. Because these agents report back to the Agent Coordinator each time they arrive and depart from a host, the Agent Coordinator can maintain information on where an agent has been and where it is going. When an agent is "lost", actions can be taken to discover where the problem occurred and what should be done as a remedy. A simple scenario will help explain the protocol.

Given: a Level 1 agent with a sequential itinerary of hosts A, B, C, and D is about to be dispatched by the Agent Coordinator. When the agent is dispatched, the Agent Coordinator begins to "track" the agent as it moves from host to host. When the agent arrives at Host A, it sends a notification message to the Agent Coordinator indicating that it arrived safely. When the agent departs for the next host, it sends another notification message indicating that it is leaving Host A with destination of Host B. Host A saves a copy of the agent to persistent memory prior to transmission. When the

agent arrives at Host B, it sends another notification message. Prior to departure, again Host B saves the agent to persistent memory and transmits a message indicating the agents departure to Host C.

Assume that some failure occurred at or before Host C. Either the agent never made it, or Host C went down just after receiving the agent, but before a notification could be sent to the Agent Coordinator. After some timeout period, the Agent Coordinator will realize that it has not yet received a notification message from Host C indicating the successful arrival of the agent. After several attempts of inquiring of Host C, all of which are unsuccessful, the Agent Coordinator contacts the last known host to successfully execute and save the agent (Host B). Host B then regenerates the agent from persistent storage and dispatches the agent to the next host in the itinerary after Host C. Upon successful arrival at Host D, a notification message is sent, informing the Agent Coordinator of its next location. This protocol continues until the agent returns to the Agent Coordinator.

Level 2 agents are equipped with medium security, in that faulty hosts are circumvented (taken care of by each agent server) and only asynchronous messaging occurs with the coordinator when needed. The protocol to handle when these agent are lost is similar to that of Level 1 agents. However, the mandatory messaging can be turned off, in which case the protocol to handle problems defaults to that used for Level 3 agents.

Level 3 agents possess the lowest form of security, in that no communication occurs with the coordinator between the dispatch of an agent and its eventual return. Care was

taken to recognize when an agent never returns. In addition, Level 3 agents may follow randomized itinerary routes if the system administrator so chooses. Since no communication takes place between the agents and the Agent Coordinator, it is impossible for the Agent Coordinator to know where an agent is at any given time.

To solve this problem, the Agent Coordinator sets a timeout period that is a function of the number of hosts to be visited on an agent's itinerary. If the timeout period expires without the return of the agent, several options are available to the Agent Coordinator. First, it can just simply notify the system administrator that the agent is late. The system administrator can then take action if necessary. Second, the agent can be dispatched a second time, with a modified timeout period to give it more time. A third option, is to increase the lost agent's security level to Level 1 and dispatch the agent again, this time tracking its activities as it migrates. This may give the Agent Coordinator information into where the problem in the network resides.

CHAPTER IV

IMPLEMENTATION

A. Introduction

In order to demonstrate the efficacy of this research, it was necessary to implement a prototype based on the design that was described in Chapter III. A secure mobile agent infrastructure was built to support experimentation. In addition, agent prototypes were developed to demonstrate their ability to be protected. Finally, intrusion detection components were constructed to demonstrate the value of using the proposed secure architecture in this domain. This chapter describes the implemented prototype by detailing how each component of the system was built. In particular, this chapter will focus on the implementation of the agent coordinator, the mobile agent servers, the agent communication system, the mobile agents themselves, and the intrusion detection components.

All components in this system are written entirely in Java version 2. Java was chosen primarily because it can be easily executed on most platforms, making the distribution of components in a heterogeneous networking environment much easier. Java was also chosen because extensive use was made of the Java Swing API which made the user interface implementation for this prototype much simpler.

B. The Agent Coordinator

The Agent Coordinator serves as the primary interface component between the system administrator and the secure mobile agent system. As such, it is primarily event-driven – that is, it awaits user input before some action is taken. The Agent Coordinator is written as a multi-threaded application because it must be able to handle several different functions at the same time. Most interactions take place using various menus, toolbars, and dialog boxes through the graphical user interface provided by the Agent Coordinator (see Figure 10). Because of the event-driven nature of the Agent Coordinator, the implementation will be described in terms of the various functions and actions that this component provides to the system administrator.

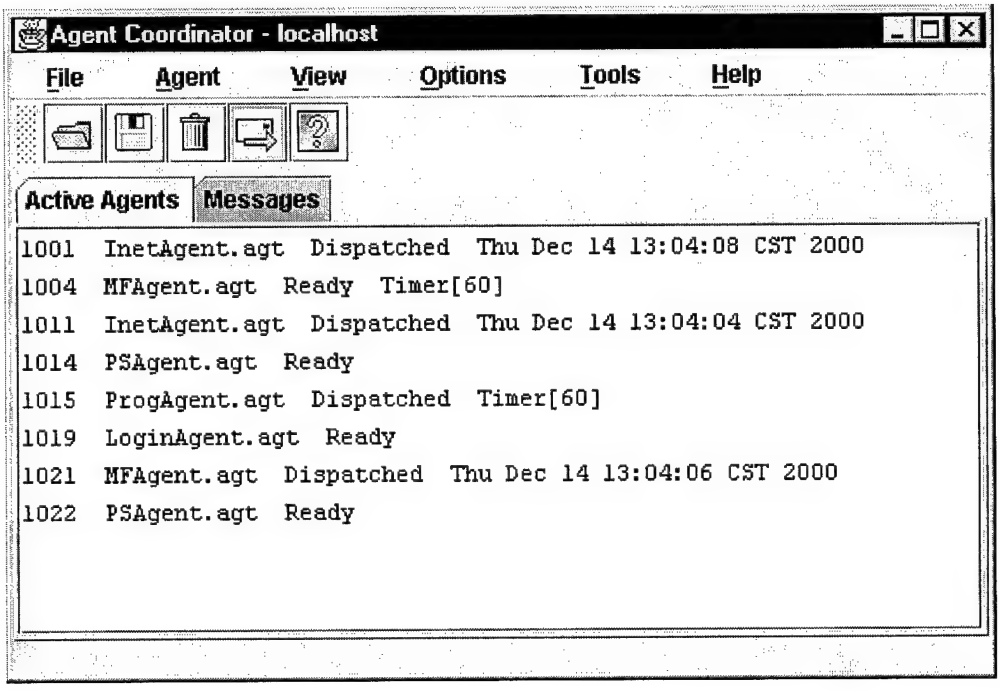

Figure 10: Agent Coordinator.

1. File Operations

The Agent Coordinator provides two primary file functions for loading and saving mobile agents to secondary storage. Because mobile agents are written in normal Java source code that must be interpreted in a Java Virtual Machine in order to execute, their existence in the form of source code is meaningless outside of such an execution environment. Therefore, the Agent Coordinator saves agents in serialized object form so that their individual identities and state can be preserved. Agent code, as well as the current state of all of its data structures is maintained on persistent storage for later retrieval. Likewise, loading a previously serialized object restores an agent to its state prior to being saved. It is interesting to note that before an agent's first use, its data structures and identity must be initialized and serialized to storage in order for the agent to be placed in a mobile and executable state.

This system uses Java's object manipulation classes to serialize objects for persistent storage. In order to save an object, a Java `ObjectOutputStream` is wrapped around a `FileOutputStream`. The agent is then written to this stream which automatically serializes the agent's code and data structures and writes the agent to a file. These serialized agent files are named with a `.agt` extension to make them easily filtered and recognizable in subsequent file operations.

The file open operation is similar in that it uses Java's object manipulation classes. In particular, an `ObjectInputStream` is wrapped around a `FileInputStream` to deserialize and load saved agents. Any potential errors with file operations (i.e. storage problems, non-existent files, etc.) are handled using Java's exception mechanism to

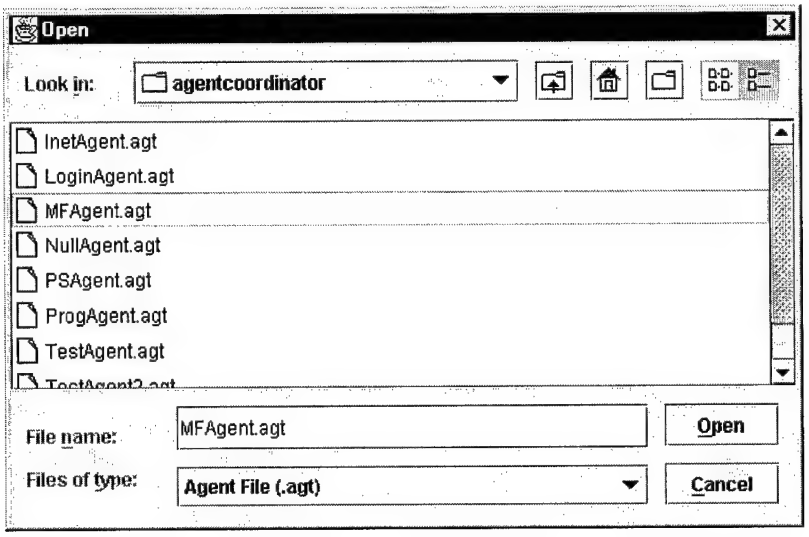

Figure 11: File dialog box.

gracefully recover and notify the user. A graphical user interface is used for all file operations to abstract away these details (see Figure 11).

2. Itinerary Operations

The Agent Coordinator maintains an internal list of all nodes in the network that are available to host mobile agents. Most agents are initialized with a pre-defined itinerary when they are loaded as specified by settings in the Agent Coordinator. The system administrator, however, can specify each agent's itinerary at any time before it is dispatched (see Figure 12). When a user chooses to change an agent's itinerary, the Agent Coordinator copies the new host information into the agent's internal itinerary data structure. This is done by overwriting the agent's itinerary `Vector` with a new `Vector` containing the new itinerary. The agent can then be saved with this new information to make it persistent, or it can serve as a temporary itinerary for the duration

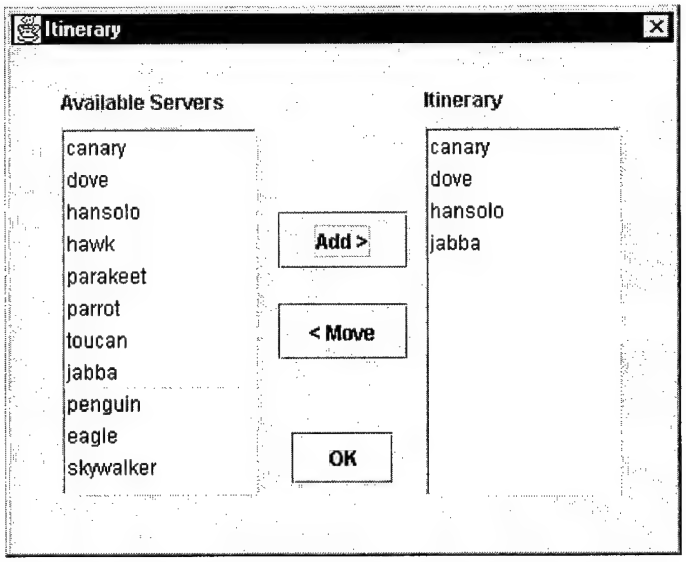

Figure 12: Manually setting agent itinerary.

of an agent's non-persistent lifetime. Based on the security level set for the agent, it will either follow the itinerary sequentially or randomly once it is mobile.

3. Agent Dispatch Function

The dispatch function is one of the primary functions of the Agent Coordinator. Many steps must occur before an agent can be successfully transmitted. The first step involves setting up a communication channel between the Agent Coordinator and the first host in the itinerary.

The Agent Coordinator queries the agent to obtain the first host in the itinerary. A socket is then opened with this remote host by connecting to its IP address on a well-known port. If the socket connection is successful, the destination host's public key is

obtained from the PKI and loaded into memory for use. The PKI provides the ability of hosts to query and obtain another host's public key.

Using this public key, an RSA encrypted `ObjectOutputStream` is setup between the Agent Coordinator and the destination host. The implementation of the RSA `EncryptStream` class is implemented in a separate security package that is described in a subsequent section. This encrypted object stream provides a secure mechanism that both transmits serialized agents as well as provides link encryption between hops.

Before the agent is actually transmitted, however, the Agent Coordinator first digitally signs the agent using the SHA-1 algorithm with the Agent Coordinator's own private key. The SHA-1 algorithm is implemented in the same security package that also provides all the other security functionality. The signed and encrypted serialized agent is then written to the open stream. Errors that might occur at any phase of this process are appropriately handled by the Agent Coordinator with appropriate error messages given to the system administrator.

4. Send Mail Function

The Agent Coordinator also provides an interface to allow the user to manually send messages to agent servers and individual agents. Using a GUI dialog box, the user can enter the destination and message content that is then formatted into proper XML format (see Figure 13). Each message is encrypted and signed by the Agent Coordinator before it is transmitted to the system's agent message-handler server. The structure and format of these messages is described in a subsequent section on secure agent communications.

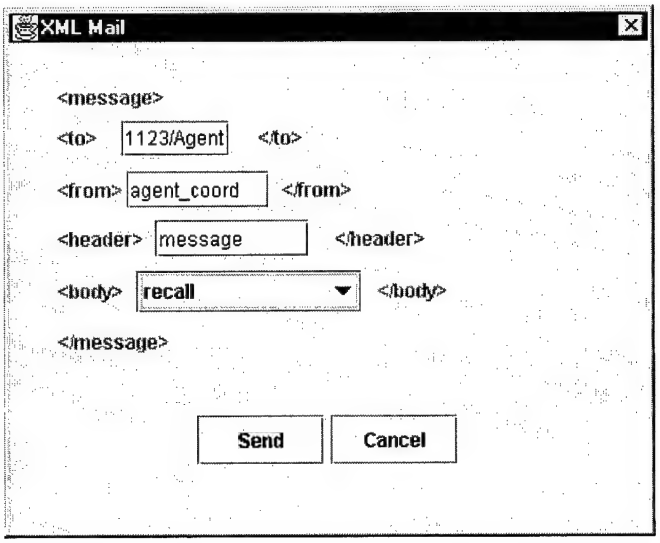

Figure 13: XML message dialog box.

5. Handling Returning Agents

In addition to dispatching agents, the Agent Coordinator must also be able to accept returning agents. The Agent Coordinator implements a listening thread that continuously listens for incoming agents on a well-known port. When this `AgentAcceptor` thread accepts a connection from an agent server, a separate `AgentHandler` thread is started that handles the processing of the incoming agent.

Because the incoming agent is returning to the Agent Coordinator, the sending host will have encrypted the agent with the Agent Coordinator's public key. Hence, the `AgentHandler` thread opens a `DecryptStream` using the Agent Coordinator's private key in order to decrypt the incoming serialized agent. When the entire agent has been read, an SHA-1 digital signature is generated using the sending host's public key. This signature is then compared to the signature carried by the incoming agent. If the

signatures do not match, the integrity of the agent has been violated and a message is generated to alert the system administrator.

If the signatures do match, the incoming agent's payload is then dumped to a file to await processing by the IDS Analysis Agent. The agent is then saved to persistent storage and all open sockets are closed. Finally, the `AgentHandler` thread terminates.

6. Monitoring Dispatched Agents

The Agent Coordinator also provides a mechanism to monitor agents as they traverse the network. This function allows the Agent Coordinator to detect agent failures and take appropriate action to remedy the situation. This is accomplished by starting a separate `AgentFollower` thread whenever an agent is dispatched.

An `AgentFollower` thread allows the Agent Coordinator to detect the two primary threats to agent availability: the loss of agents (possibly through malicious intent) and host servers that malfunction. When the `AgentFollower` thread is initialized, it is passed a copy of the agent's itinerary that it is monitoring. The thread then starts a timer that can be changed by the user to make its duration longer or shorter. Each time the timer expires, the Agent Coordinator checks for an `arrival_notification` message from the host that is next in the monitored agent's itinerary. If no message has yet arrived, the timer is restarted and the Agent Coordinator waits to again check for an incoming message. After five iterations of this process (can be changed by the user), the Agent Coordinator assumes that something has occurred to the agent and sends an agent `revive_request` to the last known host that successfully executed the agent. This

message instructs the host to restore the agent from persistent storage and to transport the agent to the host as indicated in the message. The next host is usually the server in the itinerary following that of the host that has failed to send the last notification message.

Monitoring dispatched agents is handled differently for Level 3 agents since no communication occurs with the coordinator between the dispatch of an agent and its eventual return. The only real monitoring that occurs is to take care of recognizing when an agent never returns. To monitor these type of agents, the Agent Coordinator sets a timeout period using a Java `Timer` that is a function of the number of hosts to be visited on an agent's itinerary. The value of the timer can be changed manually by the system administrator or can be updated periodically based on past performance. If the `Timer` expires without the return of the agent, the Agent Coordinator can either notify the system administrator that the agent is late, dispatch the agent a second time with a modified timeout period to give it more time, or it can increase the late agent's security level to Level 1 and dispatch the agent again, this time tracking its activities as it migrates. This may give the Agent Coordinator information into where the problem in the network resides. All of the options are user selectable.

7. Miscellaneous Functions

The Agent Coordinator provides several other functions that enhance its functionality. First, the Agent Coordinator provides the ability to add timers to individual agents. These timers allow the system administrator to indicate that an agent should be automatically dispatched by the Agent Coordinator at specified time periods.

These time periods can be seconds, minutes, hours, or day. In addition, the user can specify an exact time of day when an agent should be dispatched (see Figure 14). These timing functions are implemented using `Timer` threads that are started whenever the user selects an appropriate time interval. Each agent may have up to one `Timer` thread associated with it. When a `Timer` thread indicates that the desired interval has expired, it notifies the Agent Coordinator which subsequently dispatches the agent with no user interaction. The `Timer` thread then resets itself and begins its monitoring anew.

The Agent Coordinator also allows the user to set security related parameters for individual agents. The user may specify the security level for a given agent (low, medium, or high). These settings are stored internally with each agent so that this information is available to agent servers. Agent servers use this information to make security related decisions concerning encryption, routing, and fault handling.

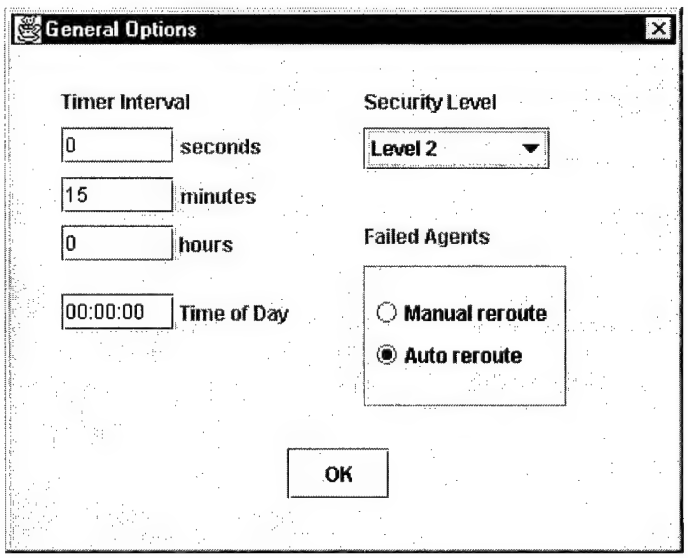

Figure 14: General options for an agent.

C. Mobile Agent Servers

Mobile agent servers provide the actual runtime environment for visiting mobile agents. An agent server must be running on each host in the network that will support mobile agent execution. In addition to providing the runtime environment for mobile agents, these servers also handle the local management, security, communications, transport, and regeneration of mobile agents.

Mobile agent servers may be executed in one of two modes. Interactive mode provides a graphical user interface to allow the system administrator to make changes to various server settings while the agent server is executing (see Figure 15). In addition, interactive mode provides substantial feedback to the system administrator that allows for monitoring and debugging of all activities taking place on a specific host. Non-interactive mode runs as a normal operating system daemon and restricts changes to settings and monitoring to only those things supported through secure messaging with the Agent Coordinator. This would be the normal operating mode of an agent server unless the system administrator had some particular interest in monitoring the activities for an individual host. This could be done by exporting the display to the system administrator who may then observe a host's behavior or make changes to its settings.

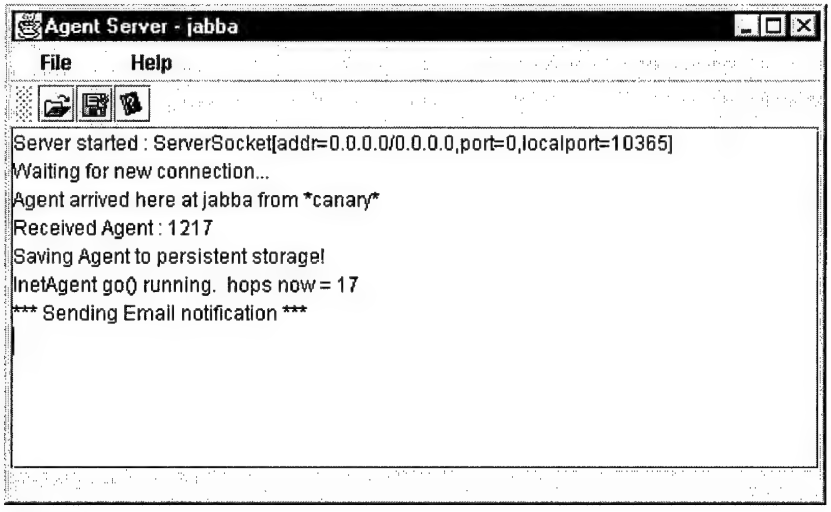

Figure 15: Agent server GUI.

Agent servers are implemented as multi-threaded applications capable of handling many agents simultaneously. The primary function of these agent servers are to wait for, accept, and execute incoming agents. The main method of an agent server implements a loop that continuously listens for incoming agents on a well-known port. Whenever an agent server accepts a connection from another host, an `AgentHandler` thread is started that handles the processing of the incoming agent.

The sending host will have encrypted the agent with the destination host's public key. Hence, the `AgentHandler` thread opens a `DecryptStream` using its own private key in order to decrypt the incoming serialized agent. When the entire agent has been read, an SHA-1 digital signature is generated using the sending host's public key. This signature is then compared to the signature carried by the incoming agent. In addition, the itinerary and code segments are checked for integrity by comparing their digital signatures using the Agent Coordinator's public key. If any of these signatures do not

match, the integrity of the agent has been violated and a message is generated to alert the system administrator. In addition, the agent is not executed, but rather is returned directly to the Agent Coordinator for checking.

Assuming that the decryption and signature checking is successful, the agent server then contacts the message server to download any messages that may be pending for the agent that just arrived. These messages are maintained in a large array that the agent may access when executed. In addition, any messages for the agent that may require the server's intervention, such as recall messages, are handled by the agent server prior to its execution. After checking the agent's security level, the agent server will send an agent receipt notification message to the Agent Coordinator for all Level 1 (high) secure agents.

The incoming agent is then executed by calling its `go()` method as an entrance into the agent. After the agent has completed its execution, the agent is then saved to persistent storage as an encrypted, serialized `Object` in case of future failure.

The agent server then prepares to send the agent to another host. For a Level 1 or Level 2 agent, the agent server chooses the next host stored in the agent's itinerary. For Level 3 agents, a host is randomly chosen from the remaining unvisited hosts in the itinerary. The agent server then attempts to open a socket and an encrypted `ObjectOutputStream` to the next host. If the connection setup is successful, the agent server encrypts and signs the agent as described previously and writes that object to the encrypted stream. If the socket setup and stream cannot be successfully established with the next host (for example, the host is down or not responding), then a failure is logged

and the agent server attempts to select the next host in the itinerary. This process continues until a successful connection is established or all possible hosts have been tried. If the agent server is not successful with any of the remaining hosts, the agent is returned to the Agent Coordinator.

In addition to checking for messages intended for a currently hosted agent, agent servers also periodically check for messages intended for itself. A shutdown message will cause the agent server to finish executing any currently hosted agents before closing all connections and ports. The server then terminates and will no longer accept mobile agents unless it is restarted. Query messages are directed at an agent server by the Agent Coordinator when particular information is requested about a server's status, such as the number of agents currently executing.

D. Secure Agent Communications

All communications in this system take place through a secure message server. This message server runs on a well-known host and port so that all components of the system have access to the communications system. The location of the message server can easily be changed, but all components must be notified so that they know which host is handling the messages.

Agents and agent servers "need knowledge about the semantics of the information" and an agreed upon protocol in order to communicate [46, 86]. All communications in this system use Extensible Markup Language (XML) formatted messages as a

standardized protocol language. XML was chosen because it is extensible, easy to format, and human-readable. Some example messages are shown in Table 1.

The message server continuously listens on a well-known port for incoming requests. When connections are established, the message server starts a new `MailHandler` thread to handle the processing of each new request. All messages are encrypted and signed by the sender using the destination server's public key and the sender's private key, respectively. As with securing agents, RSA is used for encryption and SHA-1 is used to generate the digital signatures. The message server has access to the PKI in order to access the necessary keys used to secure each message.

When an XML formatted message intended for an agent or an agent server arrives at the message server, it is parsed into its components. The `<to>` field is used to determine the message's intended destination. The message is then placed into the recipient's message box awaiting subsequent retrieval.

Message boxes are persistent message files that store all messages for a particular entity. These files consist of one or more messages in ASCII readable XML format that is stored on the message server's host drive. A separate message file is setup for each agent and agent server in the system. These files are created and initialized each time one of these components enters the system. For example, when an agent server is first started, it sends a secure XML-formatted message to the Agent Coordinator indicating that it is up and running. The message server, in response to such a message, creates and initializes a message file for the agent server.

Table 1: Sample XML messages.

```
<arrival_notification>
  <to> jabba </to>
  <from> 1001/Agent </from>
  <header> arrival_notification </header>
  <body> canary/128.194.132.192 </body>
 </arrival_notification>


<message>
  <to> jabba </to>
  <from> hansolo/128.194.135.217 </from>
  <header> message </header>
  <body> server_started </body>
 </message>

<message>
  <to> hansolo </to>
  <from> jabba </from>
  <header> system_message </header>
  <body> shutdown_server </body>
 </message>
```

When a request to retrieve messages arrives at the message server, the message box for the requesting entity is opened and read. Each message in the file that has been added to the message box since it was last checked is then sent sequentially, in order of arrival, to its destination. After each request has been serviced and all messages have been processed, the `MailHandler` thread handling the particular request closes all its connections and terminates.

E. Mobile Agents

This system implements an `Agent` base class as an `abstract`, `serializable` class in Java. This base class defines the methods and data structures that all other derived agents must implement. The reason to establish this common interface is to establish a basic form to define the components that are in common for all mobile agents. This standard agent template allows all components of the entire system to interact in the same way with any agent, regardless of its functionality.

The most important method that all agents must implement is the `go()` method. This method provides the entry point into an agent's functional code. As an agent moves from host to host, agent servers execute the `go()` method which executes the agent. For most agents, their only differences exist in their `go()` methods. As such, in order to write a new agent, the system administrator will only have to rewrite or modify an agent's `go()` method in order to perform some new function. Agents also implement other methods such as `initialize()`, `hopAddress()`, and `dump()` that perform necessary housekeeping functions such as initializing an agent's data structures, returning hosts from its itinerary, and releasing its payload.

The `Agent` base class also defines the data structures that all agents must contain. The four most critical are the `itinerary`, `payload`, `agentID`, and `sec_level`. The itinerary and payload structures are implemented as Java vectors. As their name indicates, they contain an agent's current travel itinerary and the data collected from each host. The `agentID` contains the unique agent identification number assigned by

the Agent Coordinator. The `sec_level` maintains the agent's current security level as determined by the system administrator at the Agent Coordinator prior to its dispatch.

In addition to these data structures, each agent also carries the digital signatures necessary to prove the integrity of its components to each agent server. These signatures are stored as ASCII strings, but are not visible to a potential interceptor because the entire structure of an agent is encrypted as described previously during its traversal.

Agent code is written in Java because it is easy to learn, ubiquitous, and simple. The advantages of Java are many. Because Java is interpreted, it is highly portable and easier to secure. Also, Java interpreters can be embedded in a variety of network devices. Network computing today is necessarily heterogeneous. Even in "closed" organizations, both hardware and software are often different. Thus, Java offers obvious advantages in a heterogeneous environment.

Java does have some disadvantages, but they do not greatly impact the design of the system. Programs written in Java are inefficient compared to native machine code, although just-in-time compilation mechanisms are greatly increasing the speed at which Java code is executed. Also, thread-level state cannot be saved in Java, which does not permit the resumption of execution of agents to take place at the thread-level [79]. The term "state" is typically defined as "the agent attribute values that help it determine what to do when it resumes execution at its destination" [10]. Most applications, however, do not require such fine-grained maintenance of program state across migrations. In this system, saving the high-level state of an agent by persistently maintaining the values of all of its data structures was sufficient.

Current Java-based mobile agent systems have many things in common. In addition to using the same programming language, most of them also use standard versions of the Java Virtual Machine (JVM), although a few have altered the JVM in order to implement thread-level state migration. Also, most of the current systems use Java's object serialization mechanisms in their transport protocols. These systems differ, however, in how agents are transported and the communication mechanisms that are implemented [10].

This system implements its mobile agents using Java 2.0 running on standard versions of the Java Virtual Machine that execute on Windows NT, Solaris 7.0, and Linux 2.2 platforms. This system was implemented without making changes to either the operating system or the JVMs.

F. Intrusion Detection System Analysis Engine

Upon return to the Agent Coordinator, agents dump their payload to persistent data logs. The IDS Analysis Engine, which runs concurrently with the Agent Coordinator, is responsible for analyzing these payload logs and alerting the system administrator to potential problems. Upon notification of a possible security violation, the system administrator can then take action to investigate and remedy the problem (see Figure 16).

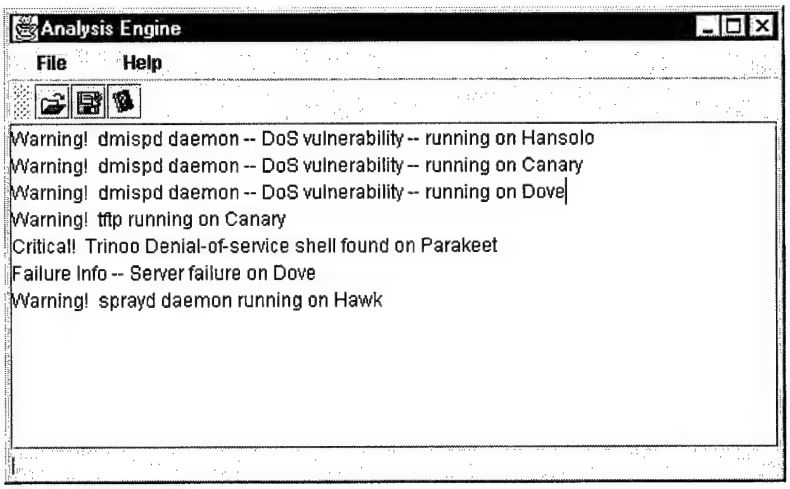

Figure 16: Analysis engine output.

The IDS Analysis Engine begins by loading a rule set for each different type of agent in the system. As agents return to the Agent Coordinator and deposit their payload, the IDS Analysis Agent compares portions of the payload against triggers in the agents' corresponding rule set. If a trigger is indicated, the actions associated with it is performed, usually involving a written notification via the terminal to the administrator for further action.

For example, one type of agent may gather information concerning which processes are currently running on each host in the network. When the agent returns the Agent Coordinator, it will dump this information. The IDS Analysis Engine, seeing a new payload from the agent, will analyze its contents by comparing each running process in the log against a set of trigger processes in the rule set. When a match is found, the IDS Analysis Agent notifies the system administrator through an alert.

Agents which collect information whose analysis is more complex than pattern matching can be handled by adding new modules to the IDS Analysis Engine. The system administrator can add additional functions that look at agent payloads for intrusive behaviors that may not be easily defined through a rule set.

G. User Interface

As stated earlier, Java was chosen primarily because it can be easily executed on most platforms, making the distribution of components in a heterogeneous networking environment much easier. Java was also chosen because extensive use was made of the Java Swing API which made the user interface implementation for this prototype much simpler. All interfaces for each component were written using standard APIs to maintain the portability and standard "look and feel" across platforms. User interfaces were written in Java 2 Platform version 1.2.

CHAPTER V

ANALYSIS AND RESULTS

A. Introduction

This chapter describes the analysis and results of this research. The process used to test and evaluate the prototype for this system is also described.

There are currently no existing systems that implement intrusion detection functionality through the use of mobile agents as described in this research. As described in Chapter II, many intrusion detection technologies exist, some of which rely on agents with very limited mobility capabilities. However, none of these systems rely on the roaming agent paradigm described in this research, and none provides security for its components that make the proposed approach feasible. Since there are no other existing equivalent systems with which to compare the methodology proposed in this research, an alternate method of evaluation was used as described in the following sections.

B. Verification

Verification is the process of ensuring that the implemented system performs as designed. This is done by comparing the conceptual design to the actual implementation of the system in a programming language. Testing each component to ensure that it accomplishes its purpose is a straightforward process. It is typical for verification to be conducted as a sequential and iterative process as components are designed and

implemented. As with other systems, this is the approach that was taken for this research. Specific details of the verification process are provided in Appendix A. Table 2 shows the number of tests performed for each component.

1. Mobile Agents

The mobile agents were the easiest components in this system to verify. Because each agent was designed with a specific purpose in mind, it was simply a matter of dispatching each agent into the network and watching its behavior in terms of movement and execution to determine if they were behaving correctly. When the agents returned, their payloads were analyzed to ensure that they were carrying the correct information.

This process can be illustrated using an example. The `PSAgent` was designed to travel to a series of hosts as determined by the system administrator, and collect a list of running processes on each host it visits. To verify it was working correctly, the agent was dispatched and its progress was monitored by watching where it migrated. Its actual migration path was compared to that given by the system administrator. When the agent returned, its payload was compared against the actual list of processes running on each host it visited. In all test cases, each agent performed as designed.

In other cases, simple "triggers" were strategically placed on various hosts in the network to verify that different agents detected problems as designed. By monitoring their progress and results, it was easily verifiable that the agent implementations did what they were designed to do.

Table 2: Number of validation tests performed.

Mobile Agents	40
IDS Analysis Engine	24
Agent Coordinator	60
Agent Servers	24

2. IDS Analysis Engine

Once the agent implementations were verified, the IDS Analysis Engine could be tested. The technique used to verify this component was to compare its results with known correct answers. That is, did the IDS Analysis Engine actually generate a detection when positive results from returning agents indicated possible intrusive behavior?

A series of agents was dispatched into the network, each looking for different signs of intrusive behavior and other system vulnerabilities. Several hosts were intentionally "seeded" with malicious indications that would be picked up by roaming agents. Upon the return of each agent, the IDS Analysis Engine was observed to see if it correctly analyzed each agent's payload. In each case, the IDS Analysis Engine gave proper indication of intrusive behavior for each type of intrusion that was planted throughout the network.

3. Agent Coordinator

The functionality of the Agent Coordinator was verified against its design iteratively as it was developed. It was observed during its execution to ensure that it functioned correctly. Because the Agent Coordinator is primarily event driven based on user input, it was relatively easy to see that the Agent Coordinator responded correctly to the user. In addition, the non-interactive functions such as the monitoring of dispatched agents and the timing of automatic dispatches were tested with a variety of scenarios to ensure that they behaved properly and as designed. To aid in the monitoring of the Agent Coordinator, a debug log was turned on to provide visibility into its functionality as it executed.

4. Agent Servers

In a similar manner, the agent servers were verified against their design by observing their functionality during execution. An optional debugging log was kept on each server to examine its behavior in handling migrating agents and agent execution. The results of agent executions were monitored to ensure that the servers were correctly executing agents when they arrived. In addition, the security functions of the servers were observed by analyzing incoming and outgoing agents to ensure that they were being properly signed and encrypted. It was also straightforward to verify that the agent servers were correctly saving agents to persistent storage and reviving them upon request by observing the persistent resources allocated to each agent server.

C. Validation

Validation is the process of ensuring that the design of the system solves the desired problem. That is, it is desirable to know how accurately the system performs its intended function. The validation of this system posed a few challenges. Since the intended purpose of the proposed system is to provide a methodology for protecting mobile agents in intrusion detection systems and to demonstrate the ability of such agents to address the shortcomings in current host-based IDSs, the validation of the system entails both determining the security of the system as well as its ability to detect intrusions when they occur. Appendix A provides specific details of the validation experiments.

1. Validating Agent Security

As described in Chapter IV, a prototype system was built to demonstrate the feasibility of this research. Using the completed prototype, many experiments were conducted for the purpose of verifying and validating the security of the entire agent architecture. The number of experiments run to validate each aspect of the agent's security is provided in Table 3.

a. Modifying Agent Itineraries

This first set of experiments involved testing the ability of the components in the system to detect when an agent's itinerary had been modified in any way. The primary purpose of these experiments was to simulate what an attacker might attempt to do if

access to an agent was obtained and what would happen if he changed the itinerary to suit a malicious purpose. For example, an attacker might change an agent's itinerary to send it off to some network "black hole" or to reroute it around a host that had already been compromised in order to escape detection.

In these experiments, the itineraries of several agents were altered after the agents had left the Agent Coordinator and were migrating "in the wild". Without fail, the next component in the system to receive such a modified agent, whether an agent server or the Agent Coordinator, detected that a modification to the itinerary had occurred. These systems immediately notified the system administrator and halted any further processing of the agent.

In another set of experiments, the itinerary of an agent was changed and a new digital signature for that itinerary was generated to replace the signature originally generated by the Agent Coordinator. Again, in each case, the next component to receive the agent recognized that the signature was a counterfeit because the Agent Coordinator's private key was not used to generate it.

Table 3: Number of agent security validation experiments performed.

Modifying Agent Itineraries	10
Modifying Agent Payloads	10
Modifying Agent Code	10
Compromising Agent Confidentiality	10
Introducing False Agents	6
Dealing with Faulty Servers	18
Lost Agents	18

b. Modifying Agent Payloads

A second set of experiments involved testing the ability of the components in the system to detect when an agent's payload had been modified. The primary purpose of this experiment was to simulate what an attacker might attempt to do if access to an agent was obtained and what would happen if he altered or deleted portions of an agent's payload. For example, an attacker might change an agent's payload to erase evidence of an intrusion in order to escape detection.

In these experiments, the payloads of several agents were altered during some part of their migration "in the wild". In each case, both agent servers and the Agent Coordinator detected that a modification had occurred. These systems immediately notified the system administrator and halted any further processing of the agent. In addition, the location of the alteration was identified, since each host has a separate payload "bay" that is signed and encrypted by the respective host.

In a another set of experiments, the payload of an agent was changed by altering the information in one bay for a particular host. A new digital signature for that host was generated to replace the signature originally generated by the host whose information was contained in that slot. The Agent Coordinator immediately recognized that the signature, and thus the data for that slot, was bogus because the respective host's public key would not yield a correct decryption.

c. Modifying Agent Code

The next set of experiments involved testing the ability of the components in the system to detect when an agent's code segment had been modified in any way. The primary purpose of these experiments was to simulate an attacker changing an agent's code for malicious purposes. An attacker might change an agent's code to render it impotent in detecting certain behaviors. Even worse, an attacker might want to change an agent's code so that it would perform some malicious act against the system it was designed to protect.

In these experiments, the code of several agents was altered after the agents had left the Agent Coordinator and were migrating "in the wild". In each case, the next component in the system to receive these modified agents detected that a change had been made to its code. These systems immediately notified the system administrator and halted any further processing of the agent.

Like the previous set of experiments, the code of an agent was changed and a new digital signature for that code was generated to replace the signature originally generated by the Agent Coordinator. In these cases as well, the next component to receive the agent recognized that the signature was bogus.

d. Compromising Agent Confidentiality

This set of experiments involved testing the ability of the system to protect the confidentiality of agents as they migrated from host to host. The primary purpose of these experiments was to simulate an attacker attempting to view an agent as it was

migrating across the network. If an intruder could see what agents were in the system and what they were doing, he might be able to evade their detection.

In these experiments, an agent was viewed as it was being transferred between hosts. In every case, the agent was indecipherable. Table 4 shows a portion of an agent as it looks when transmitted as designed using public key encryption. Table 5 shows the same portion of the agent when no encryption is used. As can be seen from Table 5, critical elements can be viewed when encryption is not used, including the agent's name, itinerary, digital signature, and other key information that might help an intruder.

As described in Chapter IV, this prototype uses the RSA cryptosystem to provide both secrecy and authentication. Its security is based on the intractability of the integer factorization problem as described in Menezes, et. al [76]. The strength and security of RSA public-key encryption has been well studied and thoroughly tested. This abundance of previous literature was relied upon in determining to use the RSA cryptosystem. The foundation of this previous research gives confidence to the security of the system.

e. Introducing False Agents

This next set of experiments involved testing the ability of the components in the system to detect when a false agent was introduced into the system to masquerade as a real agent. This simulates what an attacker might attempt to do if he were to introduce an imitation agent into the system for some malicious purpose. For example, an attacker

Table 4: Segment of an encrypted agent.

```
Ç_[±_ÄÈô¨€_råÙ_å_>]ä¯2é_Õ~X_<DÕ^ ïe¬ß5__>□}NI_€E$d/§†/Œ)\Jw"³ì__<„³üÏ…-
Îáž Ý^_hËïïƒ Â˜0Ž%÷üÉLãØF_;-Ã?¢œ_G_Ã±•L' E_-ê>s-
š£+1Ö^_Jxtó«t#ì1õ}£t_è_êaŒ_Õ¡%ÈÓY__«Ä_¢ü_‡ _¦þß(ÿ2'¿r %ýÅg$ÏÃ 'n= /Ïaä_
Z_cs nE§ÞÍ,õ
Æá¦ê¶á²_u -p_ï ø¦Ã<q¯Ex
x<úg_ ¨_j„ûMZ_!Ú_ßJÇò_s3_\3¡»€†+□RŠìu€P,ù…6GRrÏböo_ó_¬_>VŸ´_
      ªÓÖú>¾§<¥`0__ÎB¨z¶@Ý%ÛîÍ7_ãs†_C ÐHf$ü®,Øy×_:¾ÒE7>=4^ÍÂüÓ< ±ê_‡__R
ÛÈ &o_„YZOÅD¹úÚ%6K«¼&#SŒDêµü? #_#(¶=v ¹
      _¼²‰xd€_z7RhÍª^Òµùnýt Ÿ%³c_àNÁ´2_ÒgJS_3>ù_0>¥oö@HoûþWW_?Vä__?"bx_t
Û>¿ Îæ_gîÃ} Úü_5L£ÕËÃCaiŸ§X' )À®_a_ŠI;<_÷IæSCá ò?´• 3<þð©/VT_K
      êù¨X×v^Ý_È+öuYl,WÀm¯¹,nÂÁæ±[Ž€V_òr~ºèÑD$û@R3__¼ø ¶/¸R
      ½o*lStscp`,Y¾ì¨@WïS
µfÂM®i      Ù=Ë¿_mÔÒ-
~¦ÚdÙ¡>XïY"__[ ç©__³_Yõè-Ù_>__ñfÆÃ #=.@âä,5D]¹@ ,{iÙ-q õ!˜ú-
½EÄ_¼Í _B ÇªÐ\Hx^ì-"_ éznjæ_ï*k_k1G÷Ã1¯b_
Ùt__ØšSi,_:_<_§âË_ 1ƒÒÚ©+t`~u˜ÉÐ·,Ö^JÔ2HõàìJ0Ia_,Á¦Xa÷A°IS_ý95Êi|ö4ô'-
I6_«2áïÔ¿B*ÆÒTÔ/E_®,¼_*S ¯P,/
uZ-_/'¶YÄ_^¼ÎF.~f_zä__A…+®‡1h0u-sï_}ù;»)Š Óœ>_¥…_ö7òl_«û
      j,V˜"[È~¼a©à+T#m5nŸ¿ð™É ª„<'` _¯…´§Û©ç_éŠLp…ç÷p□©w¶€öë´_Ó*
™#_ª_ŸÞi˜|Öu v_íÿìºŒ_Å"ŒŸ_wÆ_kO"Mjpö8_ô%*7_Âi_º»üH( T¢o_ÓufÕ_OÝ_ÙŠª¬ú/€
psXaÒ*õïâ__€N]X_¿Nji<m',hø±´_Â Ñày"2_—_eU;4b,
```

Table 5: Segment of an unencrypted agent.

```
¬í _sr _mabids.agent.MFAgent Ýµ˜Ù_…Á_
I _agentIDI _hop_counterI _iNumberI       sec_levelI  timer_secL
      itineraryt _Ljava/util/Vector;L
itinerary_sigt _Ljava/lang/String;L _payloadq ~ _L _rndt
_Ljava/util/Random;L _visitedq ~ _xr _mabids.agent.Agent€_=ämm--_ _I
_agentIDI _hop_counterI _iNumberI   sec_levelL  itineraryq ~ _L
itinerary_sigq ~ _L _payloadq ~ _xp                pt
ËSignature(053558ec1ab34952e80cc1cb652ab754f8971be594134662b5800c699dfe
490896037853d1157deb289f5a07456b5ce80970aa8abfe905641621c1bac64e3e37,SH
A1,Fingerprint(SHA1,b068802601e8d431c750a0340bbf636bcf4aea17))p  _é
_    sr _java.util.VectorÙ—}[€;¯__ _I _capacityIncrementI
_elementCount[ _elementDatat _[Ljava/lang/Object;xp   _   _ur
_[Ljava.lang.Object; ÎXŸ_s)l_  xp
sr _java.net.InetAddress->W¯ŸãëÛ_ _I _addressI _familyL _hostNameq ~
_xp€Â„À   _t _canarysq ~ _€Â„Å   _t _dovesq ~ _€Â‡Ù   _t _hansolosq ~
_€Â‡™   _t _jabbapppppppsq ~ _   _    uq ~

ppppppppppsr _java.util.Random62-4Kð
S_ _Z _haveNextNextGaussianD _nextNextGaussianJ _seedxp
ç_4M_sq ~ _   _    uq ~
```

might create an agent that benefits him by performing data collection on his behalf and gathering information on how the system is configured.

In these experiments, rogue agents were created and were sent migrating "in the wild". In each case, the receiver of such an agent detected that an illegitimate transfer was being attempted. Because the attacker does not have access to the private keys for any of the components in the system, he cannot properly encrypt and sign these false agents. Therefore, these systems recognize an incorrectly transmitted agent and immediately notified the system administrator.

f. Dealing with Faulty Servers

The next set of experiments involved testing the ability of agents and agent servers to deal with faulty servers. The primary purpose of this experiment was to simulate what might happen if some servers happen to be down, either by chance or by malicious intent. An attacker, for example, might bring down a server or series of servers in an attempt to hide his activity.

In cases where the next server on an agent's itinerary was down prior to the start of its migration, the sending agent server also recognized the communication failure, notified the system administrator, and then routed the agent to the next server in its itinerary. In cases where multiple servers were down, the sending server keeps trying subsequent hosts in an agents itinerary until it finds one that is up. If the list of hosts is exhausted, agent servers send the agent back to the Agent Coordinator. In addition, the

Agent Coordinator, in all cases, detected when a host that was in the agent's itinerary was not visited and notified the system administrator.

g. Lost Agents

This set of experiments involved testing the ability of the components in the system to detect when an agent is lost and the ability to recover from such a situation. Agents could be lost for a variety of reasons, including hardware or software failure of a host while an agent is executing, malicious destruction by an intruder, failure of some communication mechanism in the network, etc. Regardless of the reason, the system must be able to detect this situation and correct the problem.

In these experiments, agents were subjected to loss through intentional malfeasance while they were migrating "in the wild". Agents were deleted from the system or hampered from future migrations in order to test the ability of the system to detect and correct this potential problem. In cases involving Level 1 or Level 2 agents, the Agent Coordinator detected that agents failed to report their activities after a short period of time. Following these detections, the Agent Coordinator issued a `revive_request` message to the last known server that was successfully visited by the missing agent. Once the agent was successfully revived and migrating effectively again, its progress continued to be monitored in case of future problems. In all test cases, it was not possible to delete an agent without the system quickly detected the problem. Similarly, Level 3 agents were intentionally deleted to test the system's ability to handle this

potential problem. In all cases, the Agent Coordinator notified the system administrator that missing agents had not returned within an expected amount of time.

2. Validating Intrusion Detection Functionality

One method that is often used when validating software is to compare it with other comparable applications. In the case of this research, the comparisons that were done were limited for two primary reasons. First, this prototype has no real equivalences in that it was designed with its entire architecture based on mobile agents with security mechanisms designed to make this methodology viable. No other system is designed in this way. The second reason these comparisons to other existing systems was limited, is that it was not possible to obtain, nor test against, many of the other existing IDS systems. Many commercial systems are not freely available, nor do they attempt to do what this research does. This research is unique and hence can really only be tested against its intended purpose, thus limiting the significance of using other systems for comparison.

In order to accomplish validation, however, a number of scenarios were developed which were designed to validate that the prototype does indeed protect its agents as well as detect intrusive behaviors in the intended manner. A comprehensive validation of the system's ability to detect intruders is impossible since it is not possible to know a priori all possible methods potential intruders can use to penetrate a system. Instead, the scenarios mimic common techniques used to break into system and imitate behaviors that are indicative of intrusive behavior. These scenarios are based on many of the most

common techniques used by intruders that are readily obtained from the existing literature on computer security and hacking (specific details provided in Appendix A).

a. Value Added of Mobile Agents

The roaming agent paradigm provides the ability to detect some intrusive behaviors that are not identified by other host-based intrusion detection systems. For example, certain classes of distributed attacks can be identified by agents roaming the network in ways that other systems, because of their inherent design, cannot. In addition, the roaming agent paradigm allows for detection of intrusion attempts and other anomalous activities that might raise suspicion. The roaming agent paradigm also provides the ability to detect system vulnerabilities that might make the organization easier to attack. As a consequence, these agents serve the dual purpose of both helping prevent attacks through vulnerability assessment and of detecting attacks by finding intrusive behavior.

The first class of attacks that roaming agents are uniquely designed to detect are system-wide "guessing" attacks, such as doorknob rattling. These attacks are a variant of common "guessing" attacks – those performed on a single computer that is connected to the Internet to see if it has any vulnerabilities that can be exploited. The objective of system-wide attacks, however, is to avoid detection by local host-based detection tools by spreading the probes out over multiple, independently monitored hosts. Because these attacks involve a small number of attempts, the intruder can evade detection by remaining below the threshold of detection on any given host. Even though these attacks may not register as attacks from the local host perspective, the overall organization may

indeed be under attack. These kinds of attacks are not detectable by existing host-based systems, as aggregation and correlation of activity from many different hosts is required. Since roaming agents are capable of analysis across multiple systems, however, this type of attack is easily detectable.

To illustrate, consider a normal password guessing attack. A potential intruder attempts to guess the password for a given user multiple times on the same host, hoping to eventually find the correct input. If the intruder makes too many attempts, a host-based IDS will notice the repeated login failures and indicate a possible violation. If, however, the intruder attempts to guess passwords on many machines, while ensuring that the number of guesses always stays below the threshold for login failures on any one machine, the attack will go unnoticed by conventional security tools. Roaming agents allow this type of distributed attack to be recognized and detected.

Mobile agents also provide the unique ability to detect system-wide vulnerability scanning, such as distributed port sweeps. While most host-based intrusion detection systems detect port sweeps on the individual host it is monitoring, the ability to detect sweeps below detectable thresholds occurring on multiple hosts can only be done when data is shared among hosts in the network. Roaming agents provide an effective mechanism with which to detect when potential intruders are scanning an organization's networks for possible weaknesses.

As an example, TCP scans are typically detected by host-based intrusion detection systems when a large number of ports are scanned within a short period of time. These systems flag suspicious behavior when some minimum port threshold is indicated. That

is, these systems trigger when the number of connections to destination ports on the local machine being monitored exceeds the number of connections allowed by a single remote host. The problem with this technique is that a potential intruder can successfully scan a series of ports on multiple machines in a network, each of which is below the threshold for remote connections on any one machine. This gives the would-be attacker as much, if not more information, without ever triggering that a potential attack has ever taken place. Roaming agents, however, can detect these types of distributed scans by collecting and correlating information on scans from each host to detect whether a distributed scan has taken place.

Detecting staged attacks is another unique capability of using mobile agents for intrusion detection. Staged attacks are attacks that take place in phases over time and may involve many different hosts. The recent distributed denial-of-service attacks such as Trinoo, Tribe Flood Network, and others are one form of such attacks. Typically, these attacks begin by the intruder gaining access to a legitimate account on one or more hosts. The intruder then installs additional tools to help him further the attack. Usually, a time bomb, daemon, or some other remotely controlled entity is created and is setup to wait for future commands. At some point in the future, possibly many month later, the attacker can initiate his attack by contacting these entities that were previously produced. If all of these steps were conducted on a single host or within a short span of time, conventional host-based intrusion detection systems could probably detect these attacks. Newer forms of these attacks, however, can be divided into phases over many hosts for longer periods of time, thus escaping detection by these systems. Using the roaming

agent paradigm, these forms of attacks can be detected because agents can correlate network-wide events that might indicate that one of these attacks has occurred. For example, a roaming agent could detect that a number of new daemons, all of which may have different names in order to evade detection, are listening to the same set of ports on multiple machines.

In addition to all of these advantages, the roaming agent paradigm also allows the intrusion detection system to be tailored to new threats and vulnerabilities much faster and easier than traditional host-based systems. Because new agents can be quickly written when recently discovered vulnerabilities are published, organizations using this paradigm can be protected faster than those that rely on commercial production of new patches and detection signatures, many of which take months to be published, distributed, and installed. This is a critical advantage, because most attackers exploit recently discovered vulnerabilities knowing that most systems have already been hardened against older ones. Vigilant system administrators can move to rapidly implement new security measures by creating or altering mobile agents to reduce the window of vulnerability from these attacks.

b. Comparison to Other Systems

As mentioned earlier, the validation of this system, particularly the efficacy of the roaming agent paradigm in detecting intrusions, posed a few challenges. Because no comparable systems exist, comparisons were made against other host-based intrusion detection systems that are representative of the current state of IDS technology. Table 6

Table 6: Systems used for validation.

Psionic HostSentry (Psionic Software, Inc.) [87] Psionic PortSentry (Psionic Software, Inc.) [88] Psionic LogCheck (Psionic Software, Inc.) [89] LIDS (Linux Intrusion Detection System) [90] Openwall Project IDS [91] NetSaint Network Monitor [92]

lists the systems that were obtained and tested. In addition to those that were actually acquired for testing, several other systems were considered and their architectures were analyzed from publicly available literature, but were not acquired because they are commercial products that require a fee. These systems are listed in Table 7.

Because the prototype for this research was built to show the feasibility and value of the roaming agent paradigm to intrusion detection, it does not provide complete coverage for every conceivable security vulnerability. Were this approach to be implemented as a commercial product, many more agents would undoubtedly be written to provide detection for all currently known intrusive behaviors. Hence, the validation of the system to ensure that the design of the system solves the desired problem is intended to show the additional value and functionality that secure, roaming agents provide over that of current host-based intrusion detection systems.

It is instructive to briefly describe the systems used to help validate this prototype. Psionic Software, Inc. is an Austin-based computer security company. Three of their key products, HostSentry, PortSentry, and LogCheck are designed to work together to

Table 7: Commercial security products.

CyberCop Monitor (Network Associates, Inc.) [93] Dragon Sensor (Network Security Wizards, Inc.) [94] Dragon Squire (Network Security Wizards, Inc.) [95] Patriot IDS (Patriot Technologies, Inc.) [96] PreCis Security Toolkit (PRC, Inc.) [97] Sygate Enterprise Network (Sygate Technologies, Inc.) [98]

provide better coverage of potential vulnerabilities [87]. These tools are host-based intrusion detection and vulnerability assessment products that look for problems for the host on which the reside.

HostSentry [87] is the intrusion detection component that primarily checks for login anomalies. By monitoring interactive login sessions to the host, it can detect unusual activity that might be indicative of intrusive behavior. While this function is beneficial for the host it is running on, HostSentry does not interact with other hosts to detect more sophisticated distributed attacks.

PortSentry [87] is the Psionic component designed to detect and respond to port scans against the host. This function is essential for detecting actual attacks or precursors to attacks that often occur when potential intruders are gathering information. Like HostSentry, PortSentry is a great product that does a good job detecting problems at a single host. However, this product has no capability of interacting with other hosts to detect more sophisticated forms for this attack.

Logcheck [87] is the final component to Psionic's software suite that provides the ability to process UNIX log files generated by the other components. In addition, LogCheck does a good job at detecting a variety of other operating system specific violations and other out of the ordinary system events. While LogCheck uses the email system to send reports to system administrators about the logs for the host on which it resides, it provides no mechanism to examine log files across hosts.

LIDS (Linux Intrusion Detection System) [90] is an intrusion detection system that is built into the Linux kernel and is installed as a patch. It was designed to implement those security features that are not part of the native Linux kernel, such as mandatory access control, port scan detection, and some forms of file protection. Because these features are added into the kernel, it makes turning these mechanisms off much more difficult for the potential intruder. While LIDS does an excellent job in hardening the security tools for a single host, it does not provide mechanisms to detect intrusive behaviors across hosts and thus misses many of the new, more advanced attacks. In addition, because the tools are built into the kernel, any changes to the security applications must involve a rebuild of the kernel on each system. This would be unacceptable for large organizations that intend to stay up-to-date on protecting their systems from newly discovered vulnerabilities.

Openwall [91] is similar to LIDS, in that it provides a series of patches to the standard operating system kernel. Openwall and LIDS are compatible, however, and can coexist. Openwall provides an additional set of intrusion detection tools, such as the ability to detect some forms of buffer overflow attacks, data spoofing attacks, and denial

of service attacks. While these improvements are good, Openwall suffers from the same problems as LIDS: it is unable to detect distributed attacks and requires changes to the kernel in order to be effective. Openwall and LIDS used in conjunction are very effective at hardening individual hosts, but do nothing to detect attacks across hosts in an organization.

NetSaint Network Monitor is a program [92] designed to monitor hosts and services on a network. It was written and designed to operate primarily on most Unix/Linux based platforms and can execute as a daemon to periodically run checks on user specified activities. NetSaint was designed primarily to monitor network services (such as SMTP, POP3, HTTP, etc.) and host resources (i.e. processor load, disk usage, etc.), although it can be expanded to check for other user-developed problems using a plugin interface. As with the other systems mentioned, NetSaint does an excellent job in monitoring the hosts on which it is running. It is not capable, however, of sharing and correlating distributed attacks such as password guessing and network-wide port scanning.

The first set of scenarios for validation were designed to illustrate the ability of the prototype of detect distributed "guessing" attacks that other conventional host-based tools could not. In these scenarios, a potential intruder attempted to guess the password for a given user one or two times on many different hosts on the network. Care was taken to ensure that the simulated intruder did not make too many attempts on any one host, in order to prevent the conventional host-based systems from detecting the repeated login failures on a single host which is indicative of a possible violation. Fabricated

usernames and passwords were used so as not to interfere with real users on the system. In all test cases (see Appendix A), the roaming agent prototype detected these distributed "guessing" attacks and notified the system administrator of the problem. None of the other security tools that were tested detected this type of attack and were limited to single host notifications only.

Mobile agents also provide the unique ability to detect system-wide vulnerability scanning, such as distributed port sweeps. The next set of tests were designed to show that the prototype could detect these types of attack and that the other security tools could not. In these attacks, a variety of different ports were swept across several hosts. In cases where these TCP scans attempt to open a large number of ports on a single host, all of the security tools capable of detecting port sweeps detected that a port sweep occurred on that host. In cases where the number of sweeps was low and occurred on multiple hosts in the network, only the roaming agents provided an effective mechanism with which to detect these attacks. Because the other tools did not record, nor share incidents with other hosts of "trivial" port sweeps, they were incapable of detecting these attacks.

Using mobile agents in a roaming paradigm also provides the ability to detect interesting behaviors across hosts that may be indicative of intrusion, including forms of staged attacks. To test this ability, a number of dummy daemons were created and brought up on various hosts in the network. These daemons were given different names, but were all listening on the same port. The daemons mimicked possible zombies that many distributed denial-of-service tools use in early stages of an attack. By examining

the running processes on each host, along with the network statistics for each host's ports, the prototype was able to detect that a possible problem existed on several machines in the network. By correlating the port numbers rather than the daemon names, the agents detected that new daemons were listening where none had been previously detected. None of the conventional systems could detect this type of attack.

In addition, tests were run to detect unusual resource usage across hosts in the network, indicative of a possible problem. While increased resource usage, such as enlarged CPU utilization on a single host, would not usually be anything out of the ordinary, if many systems in a network suddenly have increased usage that is sustained for a period of time, it might indicate that something is wrong. Reports have surfaced in the past of unauthorized employees farming out processing to many different machines in his organization for his own purposes, in essence stealing CPU cycles. Unless detection of this activity is shared across hosts, it is impossible to know that a problem exists. By starting a series of mathematically intensive dummy programs on each host, the CPU utilization was decreased for each machine. The prototype detected such problems by using agents to gather performance and resource usage statistics from many hosts and correlating the results with the IDS Analysis Engine. Using previous and current data, the prototype detected that a trend existed across hosts and notified the system administrator. None of the other systems detected this form of attack. Table 8 indicates how many tests were performed to validate the value-added functions provided by this approach. Details of each test are given in Appendix A.

In addition to all of these advantages, the roaming agent paradigm also allows the intrusion detection system to be tailored to new threats and vulnerabilities much faster and easier than traditional host-based systems. Adding or modifying agents is relatively simple. In comparison, all of the other tools examined often require very tedious routines to install new modules or rules to look for recently discovered vulnerabilities. In addition, many of the existing systems require changes to the operating system kernel on each machine. Most of the commercial systems require patches to be installed periodically, sometimes with long periods between update availability. This is an obvious disadvantage to the conventional approach.

Table 8: Number of intrusion detection validation experiments performed.

Guessing Attacks	24
Vulnerability Scanning	24
Staged Attacks	20
Resource Usage	10

CHAPTER VI

SUMMARY AND CONCLUSIONS

A. Summary

The purpose of this research was to develop a methodology for protecting mobile agents in intrusion detection systems and to demonstrate the ability of such agents to address the shortcomings in current agent-based, host-based IDSs. This methodology supports the defense of computer systems through a secure, mobile agent-based architecture.

The first step in conducting this research was to complete a literature survey to investigate previous research in mobile agents, computer security, and intrusion detection systems. From this survey, a new approach and methodology based on secure mobile agents was developed to address the limitations in these other systems. After the design of the system architecture was developed, a prototype was built to demonstrate the viability of this proposed approach. Finally, the implementation was verified and validated.

Mobile agents offer several advantages over conventional client-server and peer-to-peer paradigms. When implemented correctly, they reduce the overall communication traffic in the network. Since the code that makes up the functionality of a mobile agent is generally much smaller than the data to be operated on, transporting the agent rather than the data greatly reduces the communications load among hosts on the network. In addition, mobile agents allow users to quickly create specialized services by tailoring

agents to a specific vulnerability or threat. Such customization is a major asset provided by code mobility because re-tailored agents can be designed to perform new functions rather quickly using existing agents for the basic framework.

Mobile agents offer several advantages when used in the intrusion detection domain over conventional host-based methods. Because agents are more easily tailored, they can be quickly added or changed to observe new host behaviors. As new intrusive indicators or vulnerabilities are discovered, new agents can be written and dispatched to look for these problems on network hosts. In addition, agents can be very efficient if they are written to be simple and to consume as few system resources as possible. These agents can impose a lower overhead on network bandwidth and other resources than can some of the current, more centralized approaches to intrusion detection. Intrusion detection systems using mobile agents are more fault tolerant in that the components that makeup the system are moving and are not centrally located. The failure of a host does not necessarily interfere with a mobile agent. This makes such systems more resilient to subversion. A final advantage of using mobile agents in this domain is that they scale well to larger systems. As the size and demands of a network grow larger, more agents can be added to migrate through hosts looking for intrusive indicators.

Even though using mobile agents for intrusion detection offers many benefits over traditional approaches, the major obstacle of providing adequate security for the agents and the infrastructure must be eliminated before these systems can be deployed in real settings. The largest hurdle to a real-world implementation of such an agent-based system is this: agents and hosts cannot trust each other. In addition, if a host is

penetrated and the attacker gains access to an agent, he may gain access to information that will help him attack other hosts in the network and further penetrate the system. If an attacker can obtain detailed knowledge of the detection systems installed at a particular site, he will be better able to avoid its triggers; thus, it would be better to deploy an IDS whose triggers are not easily analyzable. Hence, security for these agents is critical.

This research solves these many problems by ensuring agent confidentiality, integrity, and availability through a variety of methods. In addition, the problem of mutual suspicion is solved since agents and hosts can verify the authenticity and origin of each component in the system. The confidentiality of mobile agent code and data is protected during migrations between hosts. The integrity of mobile agent code is monitored so that any changes to an agent will be detected. The integrity and confidentiality of data collected during an agent's lifetime are ensured so that subsequent hosts cannot view this data or make changes without detection. Unauthorized changes to an agent's itinerary are detected. Finally, agents are protected from denial-of-service attacks.

B. Conclusions and Significance of Research

The implemented prototype shows that this methodology is sound and feasible. The implementation demonstrates several features that are unique compared with other host-based intrusion detection systems. The principle contribution of this research is to provide a methodology for protecting mobile agents in intrusion detection systems and to demonstrate the ability of such agents to address the shortcomings in current host-

based IDSs. This methodology supports the defense of computer systems through a secure, mobile agent-based architecture. The following issues were addressed by this research:

- Integrated mechanisms can be applied for securing mobile agents in intrusion detection systems that ensure agent confidentiality and integrity through the use of cryptographic and other methods. The confidentiality of mobile agent code and data is protected during migration between hosts. The integrity of mobile agent code is monitored so that any changes to an agent will be detected. The integrity and confidentiality of data collected during an agent's lifetime is ensured so that subsequent hosts cannot view this data or make changes without detection. In addition, unauthorized changes to an agent's itinerary are detected. Authentication of both agents and hosts is provided to prevent spoofing.
- Mechanisms can be applied for securing mobile agents in intrusion detection systems that ensure agent availability. Mobile agents can be protected from denial-of-service attacks. Malicious or malfunctioning hosts that attempt to remove or suspend agents can be detected. Various hold-back, persistence, and other techniques can ensure that when agents are removed from the system by anyone other than the originator, it will be detected.
- The roaming agent paradigm provides the ability to detect intrusive behaviors that are not identified by other host-based intrusion detection systems. Certain classes of distributed attacks, including common threats such as doorknob

rattling, can be identified by agents roaming the network in ways that other systems, because of their inherent design, cannot.

- The roaming agent paradigm allows the intrusion detection system to be tailored to new threats and vulnerabilities much faster and easier than traditional host-based systems. This provides a mechanism to quickly add new detection and defense components to a system as soon as vulnerabilities are discovered, rather than having to wait for weeks before new patches or rule sets are produced by the tool's designer.
- Using secure mobile agents provides a scalable solution to intrusion detection. As the size of an organization's network grows, additional agents can be deployed to provide extended coverage.
- Secure mobile agents provide platform independence. Agents can run in heterogeneous environments without having to be rewritten for any particular hardware or operating system configuration.

C. Recommendations for Future Work

The prototype implemented for this research was never intended to be used "as is" in a real world application. While the implementation fully addresses the goals established at the beginning of this research, any project of this size has many areas that can be the subject of further and continuing research. Because the prototype was written solely as a demonstration of the feasibility of this research, a large amount of work would be required before it could be deployed as a real world application. A commercial

realization of this research would require large monetary and personnel commitments, and would probably take several man-years to complete.

1. Agent Coordinator

The first area that could benefit from future work is the Agent Coordinator. The user interface to the Agent Coordinator could be written to be more user-friendly. While more than adequate for this prototype, standardizing the interactions with the system to conform to what most user's expect from a commercial product would be helpful. For example, by adding right-click popup menus, the user could do in one step what now takes several mouse clicks. In addition, adding features such as drag and drop and color highlighting might make the interface easier to use.

In addition to these cosmetic changes, several functional changes could be made to the implementation in order to allow multiple, redundant agent coordinators to coexist in the system at the same time. These redundant agent coordinators would provide redundancy in case of failure and would create additional components for an intruder to defeat in order to be successful. The problem of a central point of failure would be eliminated, but some care would have to be taken to ensure that the agent coordinators coordinate among themselves to prevent them from duplicating efforts.

2. IDS Analysis Engine

The IDS Analysis Engine written for this research is able to analyze a small variety of possible intrusive indicators. This is sufficient to show the viability for the proposed

research methodology, but in a real-world manifestation, the IDS Analysis Engine would need to have many more modules capable of analyzing a much wider variety of activity. In addition, a richer rule set should be provided to supplement these modules in order to expand the coverage of protection offered by this system.

In order to make the system easier to use, the functionality of the IDS Analysis Engine could be integrated into the Agent Coordinator. This would provide just one user interface for the system administrator to learn and use in order to operate and monitor the system. Care must be taken, however, to leave hooks available for additional modules to be added to the analysis engine so that it remains expandable.

3. Integrated Log Analysis Tool

A system such as the one proposed in this research is not intended to be used in isolation. In a real-world environment, many security tools would be running in conjunction with the roaming agent system to provide coverage for an entire organization's infrastructure. In addition to other forms of intrusion detection, such as host-based or network-based tools, vulnerability analyzers, port scanners, and other tools would also be used. It would be ideal if an integrated log analysis tool could be developed that could take inputs from these varied security tools and perform analyses.

The advantages of such a tool are numerous. First, a centralized analysis engine would obviate the need for many different tools doing much of the same thing. In addition, analyzing outputs from many different tools could provide indicators of security problems that analyses from these outputs in isolation might not provide.

Before such a tool could be developed, however, the problem of resolving the many differing formats from all of these tools must be adequately addressed. This problem alone makes this a difficult proposition.

4. Mobile Agents

Several different types of mobile agents were written to demonstrate the feasibility of this research. While this was sufficient for the prototype, many additional agents need to be written to examine a much broader category of intrusive behavior. The task of simply writing an agent to find a published vulnerability is easy, but doing so for all of the many security vulnerabilities that exist will take some time. In addition, the rule sets for the IDS Analysis Engine will need to be updated for every new agent that is written.

In addition to adding more agents to the system, some interesting research could be done in the basic functionality of the agents themselves. It would be interesting to see what benefits could be obtained by integrating artificial intelligence algorithms and techniques into the agents. Rather than simply being reflexive, agents could be outfitted with goal-based or utility-based algorithms in making decisions about what data to collect or which hosts to visit. In addition, agents could be equipped with basic planning or reasoning capabilities in order to make decisions on their own. Research into providing agents with learning and adaptation capabilities would also be very interesting.

5. Agent Servers

Agent servers provide the basic mechanisms that allow agents to migrate and execute on hosts in the network. It might improve the functionality of the entire system if agent servers had some additional capabilities. For example, if agent servers could place restrictions on the number of resources on its host that could be used by agents, they could ensure that no one host is ever overloaded with too many agents. This would also allow servers to restrict the number of agents attempting to migrate to the host and restrict the number of agents executing at any given time. Research has already been done in this area, but the functionality would have to be added to this system to determine what impact it might have on the implementation.

In addition, it would make the system more flexible if the system administrator could choose one of several encryption protocols to use in protecting agents as they migrate from server to server. Currently, the RSA public-key cryptosystem is the only supported protocol. Giving the user more options, such as using the new AES cryptosystem, would provide application compatibility and ensure expandability in the future as new encryption algorithms are developed.

Likewise, the current implementation could be made more expandable by providing hooks to link new digital signature and hash function routines as options for the system administrator. Currently, the system uses SHA-1 as the only algorithm for generating hash codes. Adding the availability of other functions, such as MD4, MD5, or RIPEMD, would give the user greater flexibility in configuring the system for an organization's particular network.

6. Agent Communication

Communication between components in this system was done using XML. Several types of XML formatted messages were defined in order to describe the communication protocol. In addition to these messages, additional message formats could be developed in the future that provide a richer set of exchanges. XML was designed to be extensible, allowing new communication tags to be defined as needed to enlarge the universe of things that can be described.

REFERENCES

[1] W. Jansen, P. Mell, T. Karygiannis, and D. Marks, "Applying Mobile Agents to Intrusion Detection and Response," NIST Interim Report (IR) 6416, National Institute of Standards and Technology, Computer Security Division, Gaithersburg, MD, October, 1999.

[2] R. Heady, G. Luger, A. Maccabe, and M. Servilla, "The Architecture of a Network Level Intrusion Detection System," Technical Report CS90-20, Department of Computer Science, University of New Mexico, Albuquerque, NM, August, 1990.

[3] B. Mukherjee, L. T. Heberlein, and K. N. Levitt, "Network Intrusion Detection," *IEEE Network*, vol. 8, no. 3, March, 1994, pp. 26-41.

[4] M. Crosbie and G. Spafford, "Active Defense of a Computer System Using Autonomous Agents," Technical Report CSD-TR-95-008, COAST Group, Department of Computer Sciences, Purdue University, West Lafayette, IN, February, 1995.

[5] N. M. Karnik and A. R. Tripathi, "Design Issues in Mobile Agent Programming Systems," *IEEE Concurrency*, vol. 6, no. 3, July-September, 1998, pp. 52-61.

[6] D. Chess, B. Grosof, C. Harrison, D. Levine, C. Parris, and G. Tsudik, "Itinerant Agents for Mobile Computing," IBM Research Report RC 20010, IBM Research Division, T.J. Watson Research Center, Yorktown Heights, NY, March 27, 1995.

[7] K. Rothermel, F. Hohl, and N. Radouniklis, "Mobile Agent Systems: What Is Missing?," in *Proc. International Working Conference on Distributed Applications and Interoperable Systems (DAIS'97)*, Cottbus, Germany, September 30, 1997, pp. 111-124.

[8] T. Chia and S. Kannapan, "Strategically Mobile Agents," in *Proc. First International Workshop on Mobile Agents*, vol. 1219, *Lecture Notes in Computer Science*, K. Rothermel and R. Popsecu-Zeletin, Eds., Berlin, Germany: Springer-Verlag, 1997, pp. 1-10.

[9] D. Kotz and R. S. Gray, "Mobile Agents and the Future of the Internet," *ACM Operating Systems Review*, vol. 33, no. 3, August, 1999, pp. 7-13.

[10] D. B. Lange, "Mobile Objects and Mobile Agents: The Future of Distributed Computing?," in *Proc. European Conference on Object-Oriented Programming*, Brussels, Belgium, July 20-24, 1998, pp. 1-12.

[11] Y. Aridor and D. B. Lange, "Agent Design Patterns: Elements of Agent Application Design," in *Proc. Second International Conference on Autonomous Agents*, Minneapolis, MN, May 10-13, 1998, pp. 108-115.

[12] J. Bredin, D. Kotz, and D. Rus, "Market-based Resource Control for Mobile Agents," in *Proc. Second International Conference on Autonomous Agents*, Minneapolis, MN, May 10-13, 1998, pp. 197-204.

[13] M. O. Hofmann, A. McGovern, and K. R. Whitebread, "Mobile Agents on the Digital Battlefield," in *Proc. Second International Conference on Autonomous Agents*, Minneapolis, MN, 1998, pp. 219-225.

[14] A. Fuggetta, G. P. Picco, and G. Vigna, "Understanding Code Mobility," *IEEE Transactions on Software Engineering*, vol. 24, no. 5, May, 1998, pp. 342-361.

[15] N. Minar, K. H. Kramer, and P. Maes, "Cooperating Mobile Agents for Mapping Networks," in *Proc. First Hungarian National Conference on Agent Based Computing*, Budapest, Hungary, May, 1998, p. 12.

[16] W. R. Cockayne and M. Zyda, *Mobile Agents*, Greenwich, CT: Manning Publications Co., 1998.

[17] C. Schramm, A. Bieszczad, and B. Pagurek, "Application-Oriented Network Modeling with Mobile Agents," in *Proc. Network Operations and Management Symposium*, New Orleans, Louisiana, February 15-20, 1998, pp. 696-700.

[18] T. Magedanz, K. Rothermel, and S. Krause, "Intelligent Agents: An Emerging Technology for Next Generation Telecommunications?," in *Proc. Fifteenth Annual Joint Conference of the IEEE Computer Societies*, San Francisco, California, March 24-28, 1996, pp. 464-472.

[19] M. Crosbie and G. Spafford, "Defending a Computer System Using Autonomous Agents," Technical Report CSD-TR-95-022, COAST Group, Department of Computer Sciences, Purdue University, West Lafayette, IN, 1995.

[20] L. Garber, "Denial-of-Service Attacks Rip the Internet," *Computer*, vol. 33, no. 4, April, 2000, pp. 12-17.

[21] "Computer Emergency Response Team," Available at http://www.cert.org, February, 2001.

[22] D. Schoder and T. Eymann, "The Real Challenges of Mobile Agents," *Communications of the ACM*, vol. 43, no. 6, June, 2000, pp. 111-112.

[23] L. L. Kassab and J. Voas, "Agent Trustworthiness," in *Proc. ECOOP Workshop on Distributed Object Security and 4th Workshop on Mobile Object Systems: Secure Internet Mobile Computations*, Brussels, Belgium, July 20-21, 1998, pp. 121-133.

[24] J. J. Ordille, "When Agents Roam, Who Can You Trust?," in *Proc. First Annual Conference on Emerging Technologies and Applications in Communications*, Portland, OR, May 7-10, 1996, pp. 188-191.

[25] J. Riordan and B. Schneier, "Environmental Key Generation Towards Clueless Agents," in *Mobile Agents and Security,* G. Vigna, ed., Berlin: Springer-Verlag, 1998, pp. 15-24.

[26] D. S. Milojicic, F. Douglis, and R. Wheeler, ed., *Mobility: Processes, Computers, and Agents*, Reading, MA: Addison-Wesley, 1999.

[27] S. Franklin and A. Graesser, "Is It an Agent, or Just a Program? A Taxonomy for Autonomous Agents," in *Intelligent Agents III: Agent Theories, Architectures, and Languages,* J. Mueller, ed., Berlin: Springer-Verlag, 1997.

[28] C. F. Tschudin, "Mobile Agent Security," in *Intelligent Information Agents: Agent Based Information Discovery and Management on the Internet,* M. Klusch, ed., Berlin, Germany: Springer-Verlag, 1999, pp. 431-446.

[29] J. Vitek and G. Castagna, "Mobile Computations and Hostile Hosts," in *Proc. 10th Journées Francophones des Langages Applicatifs (JFLA)*, Avoriaz, France, January, 1999, p. 241.

[30] A. D. Rubin and D. E. Geer Jr., "Mobile Code Security," *IEEE Internet Computing*, vol. 2, no. 6, November-December, 1998, pp. 30-34.

[31] D. Hagimont and L. Ismail, "A Protection Scheme for Mobile Agents on Java," in *Proc. Third Annual ACM/IEEE International Conference on Mobile Computing and Networking*, Budapest, Hungary, September 26-30, 1997, pp. 215-222.

[32] B. S. Yee, "A Sanctuary for Mobile Agents," Technical Report CS97-537, Computer Science Department, University of California at San Diego, April 28, 1997.

[33] W. M. Farmer, J. D. Guttman, and V. Swarup, "Security for Mobile Agents: Issues and Requirements," in *Proc. 19th National Information Systems Security Conference*, Baltimore, MD, October 22-25, 1996, pp. 591-597.

[34] F. Hohl, "Time Limited Blackbox Security: Protecting Mobile Agents from Malicious Hosts," in *Mobile Agents and Security,* G. Vigna, ed., Berlin: Springer-Verlag, 1998, pp. 92-113.

[35] D. M. Chess, "Security Issues in Mobile Code Systems," in *Mobile Agents and Security,* G. Vigna, ed., Berlin: Springer-Verlag, 1998, pp. 1-14.

[36] G. Vigna, "Cryptographic Traces for Mobile Agents," in *Mobile Agents and Security,* G. Vigna, ed., Berlin: Springer-Verlag, 1998, pp. 137-153.

[37] L. L. Kassab and J. Voas, "Towards Fault-Tolerant Mobile Agents," in *Proc. Distributed Computing on the Web Workshop (DCW '98)*, Rostock, Germany, June, 1998, pp. 96-106.

[38] W. Jansen and T. Karygiannis, "Mobile Agent Security," NIST Special Publication 800-19, National Institute of Standards and Technology, Computer Security Division, Gaithersburg, MD, August, 1999.

[39] T. Sander and C. F. Tschudin, "Protecting Mobile Agents Against Malicious Hosts," in *Mobile Agents and Security,* G. Vigna, ed., Berlin: Springer-Verlag, 1998, pp. 44-60.

[40] T. Sander and C. F. Tschudin, "On Software Protection via Function Hiding," in *Proc. Second International Workshop on Information Hiding*, Portland, OR, April 15-17, 1998, pp. 111-123.

[41] N. M. Karnik and A. R. Tripath, "Security in the Ajanta Mobile Agent System," Technical Report RZ 2996, Department of Computer Science, University of Minnesota, Minneapolis, MN, May, 1999.

[42] M. Bellare and B. S. Yee, "Forward Integrity for Secure Audit Logs," Available at http://www.cs.ucsd.edu/~bsy/pub/fi.ps, February 9, 2000.

[43] K. Smith and R. Paranjape, "Mobile Agents for Web-based Medical Image Retrieval," in *Proc. 1999 IEEE Canadian Conference on Electrical and Computer Engineering*, Edmonton, Alberta, Canada, May 9-12, 1999, pp. 966-970.

[44] D. Rus, R. Gray, and D. Kotz, "Autonomous and Adaptive Agents that Gather Information," in *Proc. AAAI '96 International Workshop on Intelligent Adaptive Agents*, Portland, OR, August, 1996, pp. 107-116.

[45] B. Schneier and J. Kelsey, "Secure Audit Logs to Support Computer Forensics," *ACM Transactions on Information and System Security*, vol. 2, no. 2, May, 1999, pp. 159-176.

[46] K. Neuenhofen and M. Thompson, "A Secure Marketplace for Mobile Java Agents," in *Proc. Second International Conference on Autonomous Agents*, Minneapolis, MN, May 10-13, 1998, pp. 212-218.

[47] J. Baek, "A Design of a Protocol for Detecting a Mobile Agent Clone and Its Correctness Proof Using Coloured Petri Nets," Technical Report TR-DIC-CSL-1998-002, Department of Information and Communications, Kwangju Institute of Science and Technology, Kwangju, Republic of Korea, 1998.

[48] F. B. Schneider, "Towards Fault-Tolerant and Secure Agentry," in *Proc. 3rd ECOOP Workshop on Mobile Object Systems*, Jyvalskyla, Finland, June, 1997, pp. 1-14.

[49] J. E. White, "Telescript Technology: Mobile Agents," in *Software Agents,* J. M. Bradshaw, ed., Menlo Park, CA: AAAI/MIT Press, 1997, pp. 437-472.

[50] J. Tardo and L. Valente, "Mobile Agent Security and Telescript," in *Proc. IEEE COMPCON*, Santa Clara, CA, February 25-28, 1996, pp. 58-63.

[51] R. S. Gray, D. Kotz, G. Cybenko, and D. Rus, "D'Agents: Security in a Multiple-Language, Mobile-Agent System," in *Mobile Agents and Security,* G. Vigna, ed., Berlin: Springer-Verlag, 1998, pp. 154-187.

[52] D. Kotz, R. Gray, S. Nog, D. Rus, S. Chawala, and G. Cybenko, "AGENT TCL: Targeting the Needs of Mobile Computers," *IEEE Internet Computing*, vol. 1, no. 4, July-August, 1997, pp. 58-67.

[53] T. Walsh, N. Paciorek, and D. Wong, "Security and Reliability in Concordia," in *Mobility: Processes, Computers, and Agents,* D. Milojicic, F. Douglis, and R. Wheeler, eds., Reading, MA: Addison-Wesley, 1999, pp. 525-534.

[54] J. Baumann, F. Hohl, K. Rothermel, and M. Straser, "Mole - Concepts of a Mobile Agent System," *World Wide Web*, vol. 1, no. 3, July-September, 1998, pp. 123-137.

[55] H. Peine and T. Stolpmann, "The Architecture of the Ara Platform for Mobile Agents," in *Proceedings of the First International Workshop on Mobile Agents,* K. Rothermel and R. Popescu-Zeletin, eds., Berlin: Springer-Verlag, 1997, pp. 50-61.

[56] G. Karjoth, D. B. Lange, and M. Oshima, "A Security Model for Aglets," in *Mobile Agents and Security,* G. Vigna, ed., Berlin: Springer-Verlag, 1998, pp. 188-205.

[57] D. Johansen, R. van Renesse, and F. B. Schneider, "Operating System Support for Mobile Agents," in *Proc. 5th Workshop on Hot Topics in Operating Systems*, Orcas Island, WA, May 4-5, 1995, pp. 42-45.

[58] T. Sandholm and Q. Huai, "Nomad: Mobile Agent System for an Internet-Based Auction House," *IEEE Internet Computing*, vol. 4, no. 2, March-April, 2000, pp. 80-86.

[59] D. Wong, N. Paciorek, T. Walsh, J. DiCelie, M. Young, and B. Peet, "Concordia: An Instructure for Collaborating Mobile Agents," in *First International Workshop on Mobile Agents*, vol. 1219, Lecture Notes in Computer Science, Berlin: Springer-Verlag, 1997, pp. 86-97.

[60] A. O. Freier, P. Karlton, and P. C. Kocher, "The SSL Protocol, Version 3.0," Available at http://home.netscape.com/eng/ssl3/ssl-toc.html, May 11, 2000.

[61] J. Arthursson, J. Engblom, I. Jonsson, R. Mirza, G. Naeser, M. Olsson, R. Ottenhag, D. Sahlin, M. Schmid, B. Spolander, and E. Zolfonoon, "A Platform for Secure Mobile Agents," in *Proc. Second International Conference and Exhibition on the Practical Aplication of Intelligent Agents and Multi-Agent Technology*, London, England, April, 1997, pp. 109-120.

[62] G. Cabri, L. Leonardi, and F. Zambonelli, "Mobile Agent Technology: Current Trends and Perspectives," in *Proc. Associazione Italiana per l'Informatica ed il Calcolo Automatico (AICA'98)*, Naples, Italy, November, 1998, pp. 1-12.

[63] P. E. Proctor, *The Practical Intrusion Detection Handbook*, Upper Saddle River, NJ: Prentice Hall, 2001.

[64] H. Debar, M. Dacier, and A. Wespi, "Towards a Taxonomy of Intrusion-Detection Systems," Technical Report RZ 3030, IBM Research Division, Zurich Research Laboratory, Zurich, Switzerland, June, 1998.

[65] G. B. White, E. A. Fisch, and U. W. Pooch, "Cooperating Security Managers: A Peer-Based Intrusion Detection System," *IEEE Network*, vol. 10, no. 1, January-February, 1996, pp. 20-23.

[66] D. S. Alberts, "The Unintended Consequences of Information Age Technologies," Available at http://www.ndu.edu/ndu/inss/books/uc/uchome.html, December, 2000.

[67] D. E. Denning, "Protection and Defense of Intrusion," Available at http://www.cosc.georgetown.edu/%7edenning/infosec/USAFA.html, December, 2000.

[68] J. S. Balasubramaniyan, J. O. Garcia-Fernandez, D. Isacoff, E. Spafford, and D. Zamboni, "An Architecture for Intrusion Detection Using Autonomous Agents," COAST Technical Report 98/05, COAST Laboratory, Purdue University, West Lafayette, IN, June 11, 1998.

[69] D. Frincke, D. Tobin, J. McConnell, J. Marconi, and D. Polla, "A Framework for Cooperative Intrusion Detection," in *Proc. 21st National Information Systems Security Conference*, Arlington, VA, October, 1998, pp. 361-373.

[70] J. Evans and D. Frincke, "Trust Mechanisms for Hummingbird," Available at http://www.acm.org/crossroads/xrds2-4/humming.html, March 30, 2000.

[71] B. C. Neuman and T. Tso, "Kerberos: An Authentication Service for Computer Networks," *IEEE Communications*, vol. 32, no. 9, September, 1994, pp. 33-38.

[72] G. G. Helmer, J. S. K. Wong, V. Honavar, and L. Miller, "Intelligent Agents for Intrusion Detection," in *Proc. IEEE Information Technology Conference*, Syracuse, NY, September, 1998, pp. 121-124.

[73] M. Asaka, S. Okazawa, A. Taguchi, and S. Goto, "A Method of Tracing Intruders by Use of Mobile Agents," in *Proc. 9th Annual Internetworking Conference (INET'99)*, San Jose, CA, June, 1999, pp. 1-12.

[74] P. A. Porras and P. G. Neumann, "EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances," in *Proc. Nineteenth National Information Systems Security Conference*, Baltimore, MD, 1997, pp. 353-365.

[75] B. Schneier, *Applied Cryptography, Second Edition*, New York: John Wiley & Sons, Inc., 1996.

[76] A. J. Menezes, P. C. van Oorshot, and S. A. Vanstone, *Handbook of Applied Cryptography*, Boca Raton, FL: CRC Press, 1997.

[77] W. Diffie and M. E. Hellman, "New Directions in Cryptography," *IEEE Transactions on Information Theory*, vol. IT-22, no. 6, November, 1976, pp. 644-654.

[78] W. Stallings, *Network and Internetwork Security: Principles and Practice*, Englewood Cliffs, NJ: Prentice Hall, 1995.

[79] A. Corradi, M. Cremonini, and C. Stefanelli, "Locality Abstractions and Security Models in a Mobile Agent Environment," in *Proc. Seventh IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, Los Alamitos, CA, June 17-19, 1998, pp. 230-235.

[80] H. Vogler, T. Kunkelmann, and M.-L. Moschgath, "An Approach for Mobile Agent Security and Fault Tolerance Using Distributed Transactions," in *Proc. International Conference on Parallel and Distributed Systems*, Seoul, Korea, December 10-13, 1997, pp. 268-274.

[81] M. S. Greenberg, J. C. Byington, and D. G. Harper, "Mobile Agents and Security," *IEEE Communications Magazine*, vol. 36, no. 7, July, 1998, pp. 76-85.

[82] T. Thorn, "Programming Languages for Mobile Code," *ACM Computing Surveys*, vol. 29, no. 3, September, 1997, pp. 213-239.

[83] P. Felber, R. Guerraoui, and M. E. Fayad, "Putting OO Distributed Programming to Work," *Communications of the ACM*, vol. 42, no. 11, November, 1999, pp. 97-101.

[84] M. Blaze, J. Feigenbaum, and J. Lacy, "Decentralized Trust Management," in *Proc. 1996 IEEE Symposium on Security and Privacy*, Oakland, CA, May 6-8, 1996, pp. 164-173.

[85] J. Kay, J. Etzl, G. Rao, and J. Thies, "The ATL Postmaster: A System for Agent Collaboration and Information Dissemination," in *Proc. Second International Conference on Autonomous Agents*, Minneapolis, MN, May 9-13, 1998.

[86] J. Fiedler, "A Distributed Personalized News System Based on Mobile Agents," in *Proc. 36th Annual ACM Southeast Conference*, Marietta, GA, April 1-3, 1998, pp. 130-135.

[87] HostSentry, Psionic Software Inc., Available at http://www.psionic.com/, 2000.

[88] PortSentry, Psionic Software Inc., Available at http://www.psionic.com/, 2000.

[89] LogCheck, Psionic Software Inc., Available at http://www.psionic.com/, 2000.

[90] Linux Intrusion Detection System, LIDS.org, Available at http://www.lids.org, 2000.

[91] OpenWall, OpenWall Project, Available at http://www.openwall.com, 2000.

[92] NetSaint Network Monitor, Available at http://netsaint.sourceforge.net, 2000.

[93] CyberCop Monitor, PGP Security, Available at http://www.pgp.com/products/cybercop-monitor/default.asp, 2000.

[94] Dragon Sensor, Network Security Wizards, Inc., Available at http://www.securitywizards.com/intro.html, 2000.

[95] Dragon Squire, Network Security Wizards, Inc., Available at http://www.securitywizards.com/intro.html, 2000.

[96] Patriot IDS, Patriot Technologies, Available at http://patriot-tech.com/ids.htm, 2000.

[97] PreCis Security Toolkit, PRC PreCis, Available at http://www.bellevue.prc.com/precis/, 2000.

[98] Sygate Enterprise Network, Sygate Technologies, Inc., Available at http://www.sygate.com/products/sms_ov.htm, 2000.

APPENDIX A

VERIFICATION AND VALIDATION DETAILS

Verification Experiments

To verify correct implementation of design for **mobile agents**, the following actions were taken for each mobile agent written:

InetAgent
LoginAgent
MFAgent
ProgAgent
PSAgent
NullAgent
TestAgent1
TestAgent2
TestAgent3

This test involved loading the agent into the Agent Coordinator and setting its itinerary to the following: canary, dove, hansolo, jabba. Each agent was dispatched into the network, with each agent server set to debug output mode. The sequence of hops was compared against the following:

1 – arrival at canary
2 – arrival at dove
3 – arrival at hansolo
4 – return to Agent Coordinator (jabba)

Upon return, the payloads of each agent was analyzed against known results as follows:

InetAgent – contents of inetd.conf
LoginAgent – failed login attempts for each machine
MFAgent – malicious files intentionally placed on each machine (Trinoo, Tribal Flood, etc) as triggers
ProgAgent – detection of triggers – unauthorized executing programs (i.e. tftp)
PSAgent – list of currently executing processes
NullAgent – no payload
TestAgent1 – test file on each host for TestAgent1 as trigger

TestAgent2 – test file on each host for TestAgent2
TestAgent3 – test file on each host for TestAgent3

To verify correct implementation of design for the **IDS Analysis Engine**, the following actions were taken for each mobile agent written:

InetAgent
LoginAgent
MFAgent
ProgAgent
PSAgent
NullAgent

This test involved loading the agent into the Agent Coordinator with a predetermined itinerary. Each agent was dispatched into the network, with each agent server set to debug output mode. Upon return, the IDS Analysis Engine was observed for each agent as follows:

InetAgent – Warning! sprayd daemon running on (host name) for each host
LoginAgent – Warning! Possible Doorknob rattling by user: (username) on (host names)
MFAgent – Trinoo Denial-of-service shell found on (host name)
Possible Trinoo master located on (host name)
Possible Trinoo client daemon located on (host name)
Possible Tribal Flood client located on (host name)
Warning! Possible Tribal Flood daemon localed on (host name)
ProgAgent – Warning! tftp running on (host name)
PSAgent – Warning! TFTP daemon running on (host name)
Warning! dmispd daemon -- DoS vulnerability -- running on (host name)
NullAgent – no output

Sprayd was executing on each host in the network. A bogus username bill was attempted with guessed passwords on canary, dove, and hansolo. Trinoo and Tribal Flood shells were installed on canary and dove. TFTP was executed on hansolo. The dmispd daemon was executing on each host.

To verify correct implementation of design for the **Agent Coordinator**, the following actions were taken for each agent:

InetAgent – automatic timer set for 15 seconds
LoginAgent – automatic timer set for 30 seconds

MFAgent – automatic timer set for 1 minute
ProgAgent – automatic timer set for 5 minutes
PSAgent – automatic timer set for 15 minutes
NullAgent – automatic timer set for 1 hour

This test involved loading the agent into the Agent Coordinator with a predetermined itinerary. Each agent had a timer set for automatic dispatch into the network. The Agent Coordinator was observed to see if the timers worked correctly in each case for the time set. In addition, to these tests, the monitoring function of the Agent Coordinator was tested for each agent as follows (each with an itinerary of canary, dove, hansolo):

InetAgent – server on canary was stopped prior to dispatch
LoginAgent – server on dove was stopped prior to dispatch
MFAgent – server on hansolo was stopped prior to dispatch
ProgAgent – server on canary was stopped while executing ProgAgent
PSAgent – server on dove was stopped while executing PSAgent
NullAgent – server on hansolo was stopped while executing NullAgent

In all cases, agents automatically routed around servers that were stopped before they arrived. In the cases where agents were stopped while executing, the Agent Coordinator successfully notified the last known server to successfully host the agent and revive it to continue to the next host in the itinerary.

To verify correct implementation of design for the **Agent Servers**, the following actions were taken for each server and each agent:

InetAgent
LoginAgent
MFAgent
ProgAgent
PSAgent
NullAgent

Agent servers were placed in debug mode to observe their behavior as agents arrived, executed, and departed. In addition, stream sniffers were placed on the incoming and outgoing links for each server to observe the state of agents in transition. Finally, the persistent stores were observed to see if servers were successfully saving and restoring agents.

Validation of Agent Security

Modifying Agent Itineraries

Each of the following agents was preloaded with an itinerary of canary, dove, hansolo, and jabba. To test, each of the following agents were subjected to modifications of their itineraries as follows:

InetAgent – the host parrot was inserted into the itinerary
LoginAgent – the host dove was removed from the itinerary
MFAgent – the entire itinerary was deleted
ProgAgent – the itinerary was rearranged as hansolo, dove, canary, jabba
PSAgent – a random modification was made to the itinerary (by displacing one byte)
In an additional set of tests, the itineraries were changed as follows, but the digital signature for the itinerary was also changed:

InetAgent – the host parrot was inserted into the itinerary
LoginAgent – the host dove was removed from the itinerary
MFAgent – the entire itinerary was deleted
ProgAgent – the itinerary was rearranged as hansolo, dove, canary, jabba
PSAgent – a random modification was made to the itinerary (by displacing one byte)

In both sets of experiments, the modifications were made at each of the following locations for each agent:

Between the Agent Coordinator and canary
Between canary and dove
Between dove and hansolo
Between hansolo and jabba (Agent Coordinator)

Modifying Agent Payloads

Each of the following agents was preloaded with an itinerary of canary, dove, hansolo, and jabba. To test, each of the following agents were subjected to modifications of their payloads as follows:

InetAgent – the canary payload was modified
LoginAgent – the dove payload was deleted
MFAgent – the entire payload was deleted
ProgAgent – the payload was rearranged as hansolo, dove, canary, jabba
PSAgent – a random modification was made to the payload (by displacing one byte)

In an additional set of tests, the payloads as before, but the digital signature for the payload was also changed. In both sets of experiments, the modifications were made at each of the following locations for each agent:

Between the Agent Coordinator and canary
Between canary and dove
Between dove and hansolo
Between hansolo and jabba (Agent Coordinator)

Modifying Agent Code

Each of the following agents was preloaded with an itinerary of canary, dove, hansolo, and jabba. To test, each of the following agents were subjected to modifications of their code as follows:

InetAgent – a new go() method was added to the agent
LoginAgent – one line was added to the go() method
MFAgent – a line was removed from the go() method
ProgAgent – the go() method was deleted
PSAgent – a random modification was made to the code (by displacing one byte)

In an additional set of tests, the code was changed as follows, but the digital signature for the code was also changed:

InetAgent – a new go() method was added to the agent
LoginAgent – one line was added to the go() method
MFAgent – a line was removed from the go() method
ProgAgent – the go() method was deleted
PSAgent – a random modification was made to the code (by displacing one byte)

In both sets of experiments, the modifications were made at each of the following locations for each agent:

Between the Agent Coordinator and canary
Between canary and dove
Between dove and hansolo
Between hansolo and jabba (Agent Coordinator)

Compromising Agent Confidentiality

Each of the following agents was preloaded with an itinerary of canary, dove, hansolo, and jabba. To test, each of the following agents were sniffed in the following locations during migration, both using encryption and without:

InetAgent
LoginAgent
MFAgent
ProgAgent
PSAgent

Between the Agent Coordinator and canary
Between canary and dove
Between dove and hansolo
Between hansolo and jabba (Agent Coordinator)

Introducing False Agents

Each of the following agents was preloaded with an itinerary of canary, dove, hansolo, and jabba. To test, each of the following false agents were manually dispatched in the following locations:

TestAgent1
TestAgent2
TestAgent3

From the Agent Coordinator and canary
From canary to dove
From dove to hansolo
From hansolo to jabba (Agent Coordinator)

Dealing with Faulty Servers

The following actions were taken for each agent with a preloaded itinerary of canary, dove, hansolo, and jabba:

InetAgent
LoginAgent
MFAgent
ProgAgent
PSAgent

NullAgent

Agent servers were placed in debug mode to observe their behavior as agents arrived, executed, and departed. In addition, stream sniffers were placed on the incoming and outgoing links for each server to observe the state of agents in transition. The following servers were intentionally shutdown for each agent for each run:

Canary
Dove
Hansolo

Lost Agents

The following actions were taken for each agent with an itinerary of canary, dove, hansolo, and jabba:

InetAgent – Level 1
LoginAgent – Level 3
MFAgent – Level 2
ProgAgent – Level 1
PSAgent – Level 3
NullAgent – Level 2

On each test run, the agent was dispatched and the following servers were brought down in the following order on each pass as they were executing the agent:

Canary
Dove
Hansolo

Validation of Intrusion Detection Functionality

Value-Added Validation

To test the value-added of **distributed "guessing" attacks**, LoginAgent was altered to retrieve as payload all failed logins on each host visited within a specified time interval. The following was completed:

Canary – attempted failed login twice with username bill and guessed password
Dove – attempted failed login once with username bill and guessed password
Parrot – no attempted logins
Hawk – no attempted logins

Hansolo – attempted failed login once with username bill and guessed password

This was run twelve times, each time resetting the simulated loginlog.

In addition, the following was completed:

Canary – attempted failed login once with username john and guessed password
Dove – attempted failed login once with username john and guessed password
Parrot – no attempted logins
Hawk – attempted failed login once with username john and guesses password
Hansolo – no attempted logins

This too was run twelve times, run both with manual resets of loginlog and without.

To test the value-added of **distributed scanning attacks**, a ScanAgent was written to retrieve low-level scans below a specified threshold on each host visited within a specified time interval. This process was aided by modified scanlogd output. All connections were logged to file that were collected by the agent. Correlations between connection made from the same source within a specified amount of time were considered suspicious. The file contents were simulated for testing purposes for control after verifying the process worked in a non-simulated environment. The following was completed:

Canary – single scan on port 10001 from jabba
Dove – single scan on port 10001 from jabba
Parrot – no scan attempts
Hawk – no scan attempts
Hansolo – single scan on port 10001 from jabba

This was run twelve times, each time resetting the simulated scan file.

In addition, the following was completed:

Canary – single scan on port 10001 & 12001 from jabba
Dove – single scan on port 10001 from jabba
Parrot – single scan on 12001
Hawk – no scan attempts
Hansolo – single scan on port 10001 & 12001 from jabba

This too was run twelve times, both with manual resets of the scan log file and without.

To test the value-added of **staged attacks**, MFAgent was altered to look for program files installed in special locations, indicative of several staged denial-of-service attacks. The following files were planted as follows:

Canary – trin.sh
Dove – no files
Parrot – trib.c
Hawk – no files
Hansolo – td.c

This was run ten times.

In addition, the following was completed:

Canary – no files
Dove – master.c
Parrot – no files
Hawk – ns.c
Hansolo – no files

This was run ten times.

To test the value-added of **resource usage attacks**, ResourceAgent was written to periodically measure the CPU usage for each host in the itinerary. By comparing results to those gathered earlier, possible trends can be discovered. For this test, the following hosts were measured before and after several programs were started to use CPU time:

Canary – no CPU processes
Dove – CPU processes
Parrot – CPU processes
Hawk – no CPU processes
Hansolo – CPU processes

This was run ten times, each time measuring current CPU usage statistics.

APPENDIX B

SOURCE CODE

All the source code for the secure mobile agents prototype is provided on the accompanying compact disk. The compact disk was written as a standard computer data disk which should be readable on any computer. All utilities written or used specifically for this research are also included. All source code was written in Java 2.0. This code is readable in any standard text editor or Java development environment.

VITA

Jeffrey Wayne Humphries was born on August 19, 1970 in Tallahassee, Florida, but spent most of his childhood living in Perry, Florida. He graduated from Taylor County High School in 1988. He is a 1992 graduate of the United States Air Force Academy where he earned a Bachelor of Science degree in Computer Science and was commissioned as a Second Lieutenant in the United States Air Force. He has served in a number of positions in the Air Force, achieving the rank of Captain as of this date. During his time in the Air Force, he earned a Master of Science degree in Computer Science from the Georgia Institute of Technology in 1993. He has also served at the Air Force Information Warfare Center and the U.S. Air Force Academy as an instructor in the Computer Science department. In 2001, he was awarded a Doctor of Philosophy degree in Computer Science at Texas A&M University in preparation for his return as a faculty member at the U.S. Air Force Academy.

Jeffrey W. Humphries' permanent address is 5105 Roping Lane, Perry, Florida, 32347.